Universidad Carlos III de Madrid

Ingeniería Técnica en Informática de Gestión

Proyecto Fin de Carrera

Lognotify: análisis, diseño e implementación de un sistema de monitorización de ficheros de registro con notificaciones de escritorio

Autor
Guillermo Fariña Arroyo

Tutor
Alejandro Calderón Mateos

Resúmen

Pese a la creciente complejidad de los sistemas informáticos actuales y su capacidad de auto-gestión, la intervención humana sigue siendo necesaria a la hora de afrontar determinadas situaciones y problemáticas. Por ello, los sistemas operativos registran buena parte de su actividad y de los eventos que ocurren en el sistema en ficheros con el fin de proporcionar a sus administradores humanos un contexto sobre el que trabajar cuando sean requeridos.

El problema es que normalmente se acude a los ficheros de registro cuando se exteriorizan síntomas de mal funcionamiento o comportamientos anómalos, y cuando esto ocurre puede ser ya demasiado tarde para minimizar las consecuencias… o puede no llegar a ocurrir nunca en algunos casos.

Este trabajo propone una solución de monitorización continua de los ficheros de registro de sistemas GNU/Linux basada en notificaciones no intrusivas para el usuario.

Abstract

Despite the growing complexity of today's computer systems and their capacity for self-management, human intervention is still required when dealing with certain situations and problems. Therefore, operating systems record into log files much of their activity and the events that occur in the system in order to provide their human administrators a context on which to work when required.

The problem is that usually the reason to turn to the log files is when symptoms of malfunctioning or abnormal behaviors are externalized, and at that time may be too late to minimize the consequences... or it might not get to ever occur in some cases.

This work proposes a solution for continuous monitoring of GNU/Linux systems log files based on non-intrusive user notifications.

Índice de contenido

Índice de figuras

Índice de tablas

1 Introducción

1.1 Motivación

Un sistema operativo tiene que gestionar y afrontar un gran número de tareas, eventualidades y situaciones muy diversas para mantener el buen funcionamiento del sistema. Pese a la gran complejidad de los sistemas operativos de hoy, es virtualmente imposible garantizar ese buen funcionamiento, y eventualmente se dan situaciones imprevistas, difíciles de gestionar de manera automática y que requieren de intervención humana para ser resueltas.

Con objeto de facilitar dicha intervención humana y proporcionar a los administradores o usuarios la máxima información posible, los sistemas operativos suelen registrar todos los eventos considerados potencialmente relevantes para el funcionamiento del sistema en ficheros de registro (*log files*) para su posterior consulta. De esta forma, cuando la intervención humana es requerida, el administrador o usuario puede conocer con bastante precisión las actividades, eventos e interacciones que se estaban dando en el sistema mediante la consulta de dichos ficheros de registro, y actuar en consecuencia.

Sin embargo esta dinámica de acción, pese a ser suficiente en muchos casos, no es ideal en todas las ocasiones porque implica *actuar de manera reactiva a la ocurrencia de problemas, malfuncionamientos y situaciones indeseadas*. Esto es, primero ocurre el problema, y después se actúa para intentar corregirlo. Esto puede resultar insuficiente para evitar los daños ocasionados por la ocurrencia del problema una vez este ya se ha dado.

Adicionalmente, antes de actuar para corregir un problema, dicho problema debe haber sido *detectado*. En ocasiones la detección de la ocurrencia de un problema puede ocurrir demasiado tarde como para que la acción para corregir el mismo pueda evitar una mayoría de los daños que dicho problema pueda ocasionar.

La monitorización continua de dichos ficheros de registro del sistema con notificación de eventos en el momento en que se producen proporciona una posible solución ofreciendo una *dinámica de actuación proactiva*, que permite actuar de forma anticipada a la ocurrencia de problemas o sus efectos más graves a partir del conocimiento inmediato de las actividades del sistema, o en el peor caso, lograr la detección y consecuente actuación de manera inmediata.

Si bien cualquier tipo de vigilancia continuada puede resultar demasiado costosa si requiere dedicación exclusiva, un sistema de notificación lo menos intrusivo posible en combinación con técnicas como el filtrado de eventos para reducir el grado de interacción requerido por parte del administrador o usuario ofrecen una posible solución práctica al problema.

1.2 Objetivos

El objetivo principal del presente proyecto es plantear una solución para administradores de sistemas y usuarios finales que deseen disponer de un grado de control sobre la actividad de dichos sistemas suficiente para actuar de manera más proactiva en la prevención de problemas y la contingencia inmediata en el caso de que éstos ocurran, basada en la información proporcionada por el propio sistema en sus ficheros de registro.

Dicha solución, para resultar **práctica, eficiente** y **completa**, debe:

- **Ofrecer información sobre la actividad del sistema en el momento en que ésta se produzca** para facilitar una dinámica de acción más proactiva e inmediata
- **Ofrecer la información de la manera menos disruptiva e intrusiva posible** para que el proceso de monitorización no se convierta en una actividad con dedicación exclusiva
- **Ofrecer dicha información de forma remota** al sistema monitorizado, posibilitando al administrador la obtención de dicha información cuando ésta se origina sin que éste tenga que estar usando ese sistema en particular
- **Ofrecer el acceso a la información de múltiples sistemas simultáneamente** haciendo posible el seguimiento de más de un único equipo a la vez

La solución será planteada en la forma de un **prototipo funcional** de herramienta para GNU/Linux, siendo éste diseñado e implementado de manera que cumpla los objetivos anteriores.

Por lo tanto se plantean como objetivos parciales de este proyecto:

- Analizar los medios y tecnologías disponibles para acometer la tarea propuesta
- Diseñar un prototipo de sistema en base a los mismos que reúna las condiciones funcionales mencionadas con anterioridad
- Implementar dicho prototipo

1.3 Estructura del documento

Este trabajo se presenta estructurado en cinco capítulos:

1. **Introducción** (el presente capítulo), en el que se exponen las motivaciones para realizar el proyecto y se describen sus objetivos.
2. **Estado de la cuestión**, en el que se estudia el marco contextual del área tratada, los ficheros de registro en GNU/Linux, y las tecnologías y técnicas existentes que pueden ofrecer una posible solución al problema o ser utilizadas para dicho fin.
3. **Desarrollo de la solución**, en el que se describe el proceso de desarrollo del proyecto, desde la metodología seguida hasta la documentación generada durante las fases de Análisis, de Diseño y de Implementación, empezando por la especificación de requisitos y casos de uso del proyecto, pasándose después a evaluar las diferentes alternativas para el diseño de la solución, el diseño definitivo de la misma, primero a alto nivel y después de forma más detallada, y finalmente una descripción de la implementación de los aspectos fundamentales.
4. **Planificación y presupuesto,** en el que se detalla la organización temporal del proceso de desarrollo del proyecto acompañada de un desglose de los costes asociados a los recursos necesarios para completar el mismo.
5. **Conclusiones y trabajos futuros,** en el que se concluye con un repaso de los resultados obtenidos, el camino seguido y las conclusiones personales del autor, acompañados de un conjunto de sugerencias sobre cómo extender el trabajo realizado.

El documento se acompaña con materiales de referencia variados (un glosario de acrónimos, un manual de instalación y un manual de uso del software desarrollado) que pueden encontrarse al final del mismo.

En las últimas páginas se listan todas las referencias a trabajos externos mencionados o utilizados en algún punto del documento e información sobre las mismas, mediante notación numérica entre corchetes ([n]).

2 Estado de la cuestión

2.1 Marco del problema

Para contextualizar el problema es importante analizar las convenciones existentes sobre los ficheros de registro en GNU/Linux, su grado de adopción e implicaciones para el proyecto.

2.1.1 Estándar de jerarquía de sistema de archivos

El **estándar de jerarquía de sistema de archivos** [1], o FHS (del inglés *Filesystem Hierarchy Standard*), define una serie de convenciones sobre la estructura de directorios y archivos en un sistema UNIX [2]. Fue publicado por vez primera el 14 de septiembre de 1994, y la última versión (3.0) fue publicada el 3 de junio de 2015 por LSB (Linux Standard Base) Workgroup.

El FHS describe una jerarquía de directorios y archivos para un sistema UNIX que comienza en un *directorio raíz* (representado por la ruta "/") a partir del cual se desarrolla el resto de la estructura: por ejemplo, los directorios */bin*, o */usr*, que están contenidos directamente bajo el directorio raíz (*/*), o los directorios */usr/bin*, o */usr/lib*, que su ubican bajo el directorio */usr*. El FHS describe un propósito y unos requisitos para cada directorio y archivo propuesto.

Como parte de la estructura propuesta, el FHS define un directorio específicamente para contener archivos de *datos variables*, como datos administrativos y de registro, o archivos transitorios y temporales, con el nombre de */var*.

La estructura del directorio */var* está detallada dentro del documento de especificación del FHS, y uno de los directorios de éste, */var/log*, está destinado a contener *"ficheros de registro variados. La mayoría de registros deben ser escritos a este directorio o un subdirectorio apropiado"*.

Es decir, según la especificación del FHS, la jerarquía del sistema de ficheros para un sistema UNIX **contempla una ubicación específica para los archivos de registro del sistema y aplicaciones (*/var/log*)**, donde (ya sea directamente o en subdirectorios) estos deben residir.

Aparte del directorio en el que deben ubicarse los ficheros de registro, el FHS describe únicamente 3 ficheros de registro que deben existir y residir en */var/log*:

- *lastlog*: registro del último login de cada usuario.
- *messages*: mensajes de sistema provenientes de **syslogd**.
- *wtmp*: registro de todos los login y logout.

Esto significa que el FHS **no** proporciona un listado exhaustivo de los ficheros de registro que debe crear y mantener un sistema UNIX (ni por supuesto las aplicaciones que se ejecuten en el mismo).

2.1.2 GNU/Linux y el FHS

Como se menciona en el apartado anterior, el FHS es un conjunto de *convenciones* sobre la estructura de directorios y ficheros de un sistema UNIX, pero su aplicación no es forzosa para dichos sistemas. Esto significa que **no existe garantía** de que un sistema determinado siga estas convenciones, o que lo haga en su totalidad.

Cada distribución de GNU/Linux puede por tanto seguir en mayor o menor medida estas indicaciones, pero en la práctica la mayoría de ellas mantienen la conformidad con dicho estándar a modo de política que garantice (o al menos facilite) la máxima compatibilidad.

De las 5 distribuciones de GNU/Linux más populares según *DistroWatch* [3] (Mint, Debian, Ubuntu, OpenSUSE y Fedora), sólo OpenSUSE en su versión 10.2 y Ubuntu (versiones 6.06, 8.04 y 9.04) están certificadas por su conformidad con el estándar LSB (que incluye el FHS), y no en su última versión (5.0) sino en versiones anteriores (3.1 a 4.0) [4].

No obstante, en lo referente al FHS todas ellas mantienen una estructura de directorios en su mayor parte compatible, y **todas ellas incluyen el directorio *var/log*** como ubicación para los principales ficheros de registro del sistema y aplicaciones según documentación oficial y/o páginas de soporte [5][6][7][8][9].

Un ejemplo claro de sistema GNU/Linux que se desvía por completo del estándar FHS (de hecho, es parte importante de la razón de ser del mismo) es *GoboLinux* [10].

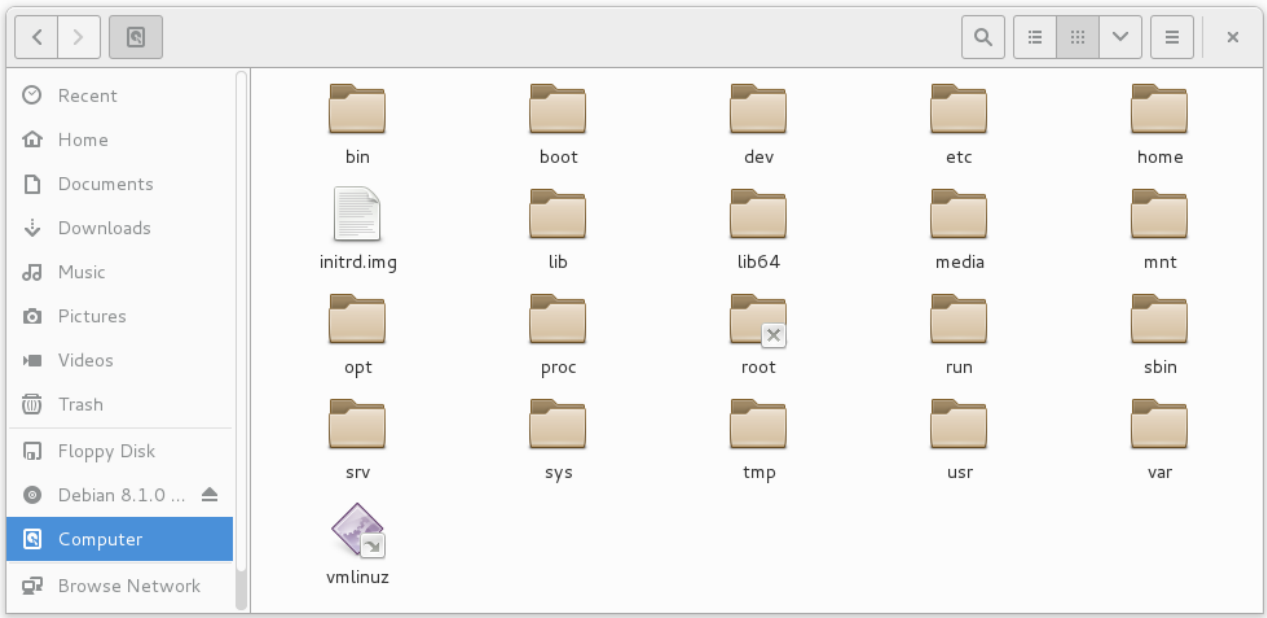

Ilustración 1: Directorio raíz en Debian 8.1 Jessie

2.1.3 Ficheros de registro en GNU/Linux

Si una ubicación estandarizada de los ficheros de registro en distintas distribuciones de GNU/Linux no está garantizada (aunque por regla general sea así), los propios ficheros de registro que puedan encontrarse en esta dirección varían considerablemente de una distribución a otra, o incluso de un sistema a otro.

Para empezar, como se menciona en el apartado 2.1.1, el FHS hace referencia únicamente a 3 ficheros de registro bajo */var/log* (*lastlog*, *messages* y *wtemp*), lo que quiere decir que el estándar no proporciona ninguna guía adicional para el resto de registros de sistema. De esta forma, distintas distribuciones incluyen registros básicamente equivalentes con nombres diferentes: por ejemplo, Debian y otras distribuciones basadas en Debian, como Ubuntu, incluyen el fichero *auth.log* que contiene registros de eventos de autenticación del sistema (por ejemplo, solicitudes de elevación de privilegios con los comandos *su* o *sudo*) [8]. Otros sistemas incluyen un fichero equivalente a este llamado *secure*.

En la mayoría de sistemas GNU/Linux **la gestión de los ficheros de registro es llevada a cabo por un demonio a tal efecto**. Por regla general, éstos son configurables, y consultando los ficheros de configuración de los mismos puede averiguarse el nombre de los ficheros en los que insertará los registros de cada tipo de actividad del sistema. El problema es que diferentes sistemas pueden utilizar diferentes demonios de registro por defecto (cada uno con sus ficheros de configuración diferentes), y estos además pueden ser reemplazados por otros diferentes por los administradores del sistema en cuestión… ejemplos de tales demonios de registro usados frecuentemente son *syslogd [11]*, *rsyslogd [12]* y *syslog-ng [13]*.

Mantenimiento y rotación de ficheros de registro

Otra consideración importante en relación al uso de ficheros de registro en el entorno actual de GNU/Linux es la gestión y mantenimiento de los mismos. En un fichero de registro se guardan entradas referentes a distintos eventos de actividad del sistema o sus aplicaciones: cada vez que se produce un evento que genera un nuevo registro en uno de estos ficheros, dicho fichero aumenta de tamaño. Esto quiere decir que sin un cierto mantenimiento, dichos ficheros pueden crecer de forma ilimitada, y según la configuración del sistema, en algunos casos muy rápidamente.

Para evitarlo, normalmente se realizan tareas de mantenimiento en los ficheros de registro, como la **rotación de los mismos**: periódicamente (con frecuencia diaria, semanal...) el fichero de registro es reemplazado por uno nuevo vacío en el que se comienza a escribir a partir de ese momento. Un cierto número de ficheros antiguos rotados se mantienen para consulta

posterior con una modificación al nombre, y los más antiguos pueden ser incluso comprimidos para que ocupen menos, o directamente eliminados. Para realizar estas tareas de mantenimiento se suele recurrir a herramientas específicamente programados para ello, como por ejemplo *logrotate*.

Formatos de ficheros de registro

Los ficheros de registro pueden tener diferentes formatos en función del programa que los crea (el demonio de registro del sistema en la mayoría de los casos, aunque distintas aplicaciones pueden crear y escribir en ficheros de registro sin intermediario) o del registro en cuestión.

Ilustración 2: Ejemplo de contenido del fichero auth.log y su formato en Debian 8.1 Jessie

Algunos de estos ficheros de registro no tienen un formato legible por personas, pues están directamente enfocados a ser interpretados por el mismo programa que los produce u otros programas diferentes. Por supuesto, para un sistema de *monitorización con notificaciones de escritorio* para usuarios, no tiene mucho sentido seguir estos ficheros cuyos registros el usuario no podría interpretar.

2.2 Soluciones existentes

A día de hoy existen ya herramientas diseñadas con objetivos similares a los propuestos, que podrían ser utilizadas como posible solución al problema planteado. Una evaluación de sus características y limitaciones servirá para dilucidar si alguna de ellas presenta una solución completa al problema propuesto.

2.2.1 tail y tailf

La solución más sencilla para la monitorización continua de ficheros de registro es utilizar los programas de consola de comandos *tail* o *tailf*.

El programa *tail* se utiliza para mostrar las últimas 10 líneas de un fichero. Mediante la opción *-f* o *--follow*, se puede solicitar al programa que muestre el contenido añadido al fichero conforme éste vaya creciendo, proporcionando así la funcionalidad de monitorización de cambios en el fichero.

Por su parte, *tailf* ejerce una función similar a *tail -f* siendo la principal diferencia según la página de manual (man, [14]) del mismo que "no accede al fichero cuando éste no crece", ahorrando los ciclos de reloj correspondientes a la consulta proactiva periódica (*polling*) del estado del fichero, y de forma colateral, evitando modificar la fecha de último acceso del mismo hasta que se produzca un cambio.

En realidad, las implementaciones modernas de *tail* recurren por defecto a notificaciones de alteración del fichero por parte del sistema de ficheros siempre que éstas estén disponibles, como por ejemplo la implementación de GNU *coreutils* [15] que puede encontrarse en las instalaciones actuales de GNU/Linux y que hace uso del subsistema de kernel *inotify*, y sólo retornan a la vigilancia proactiva del fichero cuando éstas no están disponibles.

El principal problema de *tail* y *tailf* como sistemas de monitorización de ficheros de registro por sí mismos es la carencia de funcionalidades avanzadas como la monitorización remota, la forma de notificación al usuario (que se limita a mostrar las notificaciones en consola de comandos), o el filtrado de las notificaciones resultantes para mostrar al usuario sólo una parte de las mismas.

Si bien es cierto que ambos podrían ser utilizados en combinación con otros programas y *scripts* como punto de partida para crear un sistema más complejo que posibilitase estas funcionalidades adicionales, por sí solos son insuficientes para cubrir la mayoría de objetivos declarados en la proposición del problema.

2.2.2 swatch

swatch o *Simple Log Watcher* [16] es un programa de monitorización de actividad en ficheros de registro que se ejecuta como demonio, y puede ser configurado para mostrar avisos ante ciertos eventos en forma de salida de texto en la consola, mensajes enviados a una terminal de usuario, o al correo electrónico. Qué eventos son notificados al usuario se determina mediante reconocimiento de patrones en los eventos, de forma totalmente configurable por el usuario.

Aunque en función de cómo el sistema gestione los mensajes a la terminal estos podrían ser mostrados mediante notificaciones de escritorio emergentes, el programa en sí no dispone de esta funcionalidad, y depende de cómo actúe el sistema en estos casos. Por el mismo motivo, no ofrece tampoco ningún control para configurar cómo estas notificaciones son mostradas al usuario de forma independiente al resto del sistema.

Asimismo, sus posibilidades de monitorización remota (sin combinarlo con otros programas como parte de un sistema más complejo) son limitadas, ya que los avisos al correo electrónico no son prácticos más que para avisos específicos de escasa frecuencia, pero no facilitan un seguimiento más continuado del/los sistemas monitorizados. Además, requiere acceso al sistema que se desea monitorizar para configurar quién y cómo debe recibir las notificaciones.

2.2.3 Logwatch

Logwatch es una herramienta de análisis de registros del sistema que genera informes a partir de los ficheros de registro centrándose en aquellas áreas especificadas por el usuario [17].

Los informes pueden generarse de forma automática de forma periódica, pero a diferencia de *swatch*, esta herramienta carece de la capacidad de notificación al usuario sobre la marcha (en el momento en que los eventos se producen), lo que supone al usuario tener que revisar activamente los informes cuando estos sean generados y no permite tener constancia de los diferentes eventos hasta transcurrido un tiempo después del hecho y no en el momento en que se produce.

Los informes generados pueden mostrarse como salida de texto en la consola de comandos, guardarse en un archivo para su consulta posterior, o enviarse por correo electrónico, lo que permite la monitorización remota del sistema. Teniendo en cuenta la forma de trabajo de *Logwatch* mediante generación de informes programados periódicos, el correo electrónico puede considerarse un medio perfectamente eficaz para esta tarea.

2.2.4 Tabla comparativa

A continuación se presenta una tabla comparativa de las características de las soluciones existentes previamente analizadas a modo de resumen:

	tail -f	tailf	swatch	logwatch
Informe automático				
Aviso de eventos en el momento				
Sin necesidad de realizar *polling*				
Notificaciones en consola				
Notificaciones de escritorio				
Filtrado de notificaciones				
Monitorización remota				
Notificaciones remotas				
Permite conectar sin acceder al equipo monitorizado				

Característica disponible
Característica disponible de forma parcial o con limitaciones
Característica no disponible

Tabla 1: Tabla comparativa (soluciones existentes)

2.2.5 Conclusión

Ninguna de las soluciones presentadas previamente es suficiente por sí misma para cubrir la totalidad de los objetivos propuestos.

Una de las principales motivaciones para la realización de este proyecto es la utilización de un medio de notificación no intrusivo, que permita al usuario recibir avisos sobre actividad en los ficheros de registro del sistema (y por tanto del sistema en sí) cuando esta se produce sin requerir de éste que interrumpa su flujo de trabajo para atender la notificación. Ninguno de estos programas ofrece directamente ningún medio para conseguirlo. De forma indirecta, y en función de cómo el sistema en el que se haya instalado se comporte a la hora de transmitir mensajes a la terminal al usuario, *swatch* podría ofrecer dicha funcionalidad de forma limitada, pero dependiente por completo del sistema y sin posibilidad de configuración o manipulación desde el propio programa (ya que en realidad no es una funcionalidad propia).

Otra de las características en la que las presentes soluciones se quedan más cortas es en la monitorización remota. Tanto *swatch* como *Logwatch* ofrecen la posibilidad de enviar los avisos a una dirección de correo electrónico, pero este medio es difícilmente ideal para el seguimiento continuo y en cualquiera de los casos, requiere de la revisión activa del informe enviado por parte del usuario.

Aparte de estos problemas comunes, *tail* y *tailf* son en general herramientas demasiado básicas para el tipo de uso propuesto (aunque precisamente por este motivo son una opción a tener en cuenta para crear un sistema más complejo utilizándolas como punto de partida), y *Logwatch*, pese a ser una herramienta de análisis y diagnóstico muy completa, es la única del conjunto que tampoco ofrece avisos en el momento, otro de los objetivos principales (permitir al usuario tener conocimiento de eventos de sistema que puedan requerir de una respuesta inmediata en el momento, y no transcurrido un tiempo indeterminado después del hecho).

2.3 Técnicas y tecnologías disponibles

En la actualidad existen múltiples técnicas y tecnologías disponibles para realizar o facilitar tareas similares a las que debe llevar a cabo el sistema a desarrollar, que pueden ser utilizadas como recursos de desarrollo o modelos de referencia durante el proceso si sus características se adaptan a las necesidades del mismo.

Según los objetivos planteados, el sistema debe de realizar tres tareas básicas:

- El seguimiento de los ficheros de registro en los sistemas que se desee monitorizar.
- La notificación de eventos al usuario de una forma que no resulte intrusiva para el mismo.
- La transmisión de dichos eventos de los equipos monitorizados a los equipos utilizados para realizar el seguimiento.

A continuación se proporciona un breve estudio sobre diversas tecnologías disponibles para cada una de estas tres tareas y sus características fundamentales.

2.3.1 Monitorización de ficheros

E/S de ficheros POSIX

Los sistemas basados en GNU/Linux contienen librerías de llamadas al sistema basadas en el estándar POSIX de interfaces para sistemas operativos [18] que permiten la interacción con ficheros y su entrada y salida (E/S).

El estándar POSIX define un modelo de E/S de ficheros mediante descriptores numéricos asignados por el sistema operativo para referenciar dichos ficheros, así como una interfaz basada en funciones que permiten la apertura, cierre, lectura y escritura de ficheros en diversos modos.

Estas librerías de sistema POSIX son un elemento básico en la interacción y E/S de ficheros en sistemas GNU/Linux a bajo nivel, pero no ofrece ninguna funcionalidad específicamente dirigida a la monitorización de los mismos.

Es perfectamente posible en cualquier caso implementar dicha funcionalidad a bajo nivel sobre estas funciones de sistema, comprobando periódicamente el estado de los ficheros a monitorizar en busca de cualquier cambio desde la última comprobación y obteniendo dichos cambios en el momento de producirse éstos.

Esta técnica, conocida como *polling*, es *eficaz* como solución al problema, pero no *eficiente*, ya que supone realizar constantes llamadas al sistema sólo para comprobar que no ha habido cambios.

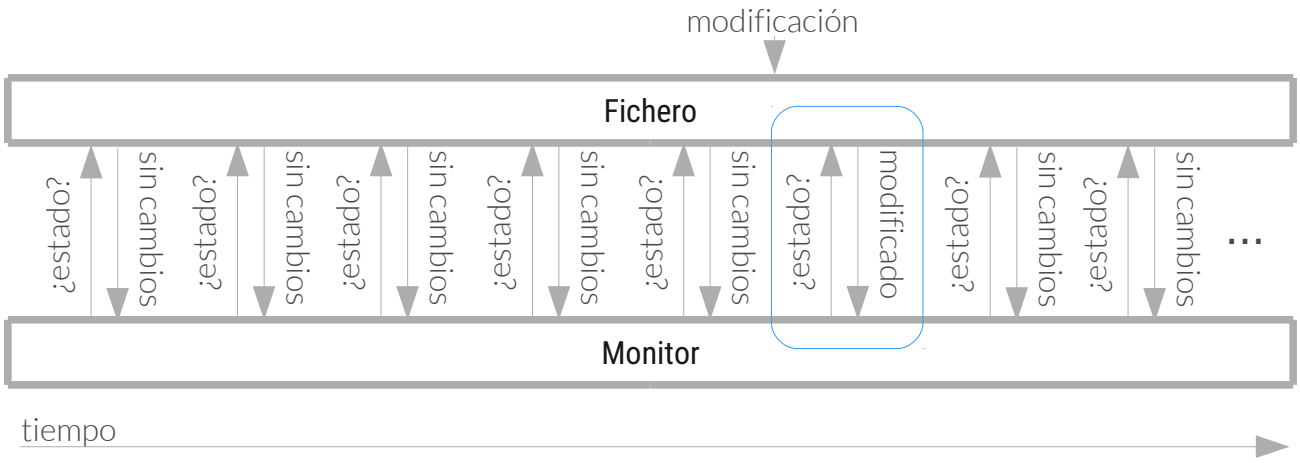

Ilustración 3: Monitorización del estado de un fichero mediante polling

tail -f y tailf

Estos programas de consola de comandos ya han sido mencionados previamente en 2.2.1, al ser contemplados (y descartados) como posible solución completa alternativa a la propuesta en el presente proyecto. Sin embargo, la funcionalidad que ofrecen merece ser contemplada como parte de una solución más compleja, en este caso, proporcionando una solución ya implementada al problema de saber cuándo un fichero ha sido modificado.

Tanto las implementaciones modernas de *tail* como *tailf* ofrecen la posibilidad de seguir a un fichero y ser notificado de los cambios producidos en el mismo mediante notificaciones por parte del sistema de ficheros (siempre que esta posibilidad esté disponible) sin tener que recurrir a la técnica de *polling* descrita en el apartado anterior.

Esto proporciona una solución eficiente y sencilla de utilizar, al menos desde la consola de comandos. La limitación principal de esta solución es el hecho de que tanto *tail* como *tailf* son programas completos, y por tanto ejecutados como procesos independientes lo que supone una serie de trabas en cuanto a la sencillez y eficiencia del tratamiento automático de esta información una vez obtenida al utilizarlos como parte de un sistema más complejo.

Tienen la ventaja adicional de estar disponibles en todos los sistemas GNU/Linux, sobre todo en el caso de *tail*, que se encuentra preinstalado como parte del conjunto de utilidades del sistema de GNU *coreutils*.

dnotify e inotify

dnotify es un monitor de eventos del sistema de ficheros incluido en el *kernel* de GNU/Linux como parte de la llamada de sistema *fcntl* [19]. Fue introducido por vez primera en la versión 2.4 del *kernel* de Linux, con el objetivo de notificar cambios en el sistema de ficheros permitiendo así la monitorización de ficheros y directorios contenidos en el mismo.

Sin embargo, *dnotify* utiliza *polling* para saber cuándo han ocurrido dichos cambios. Al no ser ésta una técnica eficiente, fue sustituido en la versión 2.6.13 del *kernel* de Linux por *inotify* (inode notify)[20], que actúa como una extensión de los sistemas de ficheros para proporcionar la misma funcionalidad sin *polling*. *dnotify* permanece accesible por motivos de compatibilidad pero la existencia de *inotify* deja efectivamente obsoleto para la mayoría de usos al anterior.

inotify ofrece una solución efectiva y eficiente a la monitorización de ficheros, permitiendo registrar "vigilancias" (*watches*) para ficheros y directorios individuales en una instancia del mismo, para posteriormente obtener mediante una llamada bloqueante eventos de cambios en los ficheros y directorios registrados tan pronto como éstos se produzcan.

inotify notifica eventos de acceso, cambios en los metadatos, de cierre, de eliminación, de modificación, de renombrado, y de apertura para los ficheros o directorios seguidos o de ficheros contenidos en el directorio seguido.

Esto proporciona una base flexible para las funciones de monitorización, y es de hecho la solución empleada por la implementación de la función de seguimiento de *tail -f* de GNU en sus *coreutils* [15].

Sin embargo, no está exento de limitaciones: no proporciona notificaciones correspondientes al contenido de subdirectorios de un directorio seguido de forma recursiva, y cada *watch* que sea registrado requiere de un descriptor de fichero asignado por el sistema, por lo que en situaciones complejas puede suponer alcanzar el límite de descriptores disponibles para el proceso que lo usa.

Además, al actuar como extensión del sistema de ficheros, hay sistemas de ficheros virtuales y especiales que pueden dar problemas de compatibilidad, como en el caso de *sysfs* y *procfs*.

2.3.2 Notificación no intrusiva de eventos al usuario

Growl

Growl es un sistema de notificaciones de escritorio desarrollado para Mac OS X [21]. Se basa en una aplicación que una vez instalada, se ejecuta en segundo plano y muestra al usuario notificaciones enviadas por otras aplicaciones que se comunican con ella.

Dichas notificaciones se muestran (por defecto) en pequeñas ventanas emergentes ubicadas bajo la esquina superior derecha de la pantalla, que permanecen un corto periodo de tiempo a la vista antes de desaparecer si el usuario decide ignorarlas.

La aplicación *Growl* que muestra las notificaciones es totalmente configurable, con diversos estilos visuales (una selección de ellos vienen incluidos en la instalación por defecto, pero es posible descargar o incluso crear otros diferentes) y opciones de visualización y uso. Estas opciones de configuración se ajustan en la aplicación que muestra las notificaciones (la aplicación *Growl*) y no desde las aplicaciones que lo utilizan, de forma que todas las notificaciones se muestran de idéntica forma independientemente de su origen.

Además, ofrece funcionalidades adicionales, como la posibilidad de que la aplicación *Growl* reciba las notificaciones de forma remota desde otros equipos.

Growl es una aplicación comercial que debe de ser comprada por el usuario para ser utilizada. Además, este sistema no es multiplataforma, y debido a este hecho es únicamente compatible con OS X. Existen otras versiones alternativas para sistemas operativos adicionales, como *Growl for Linux* [22] y *Growl for Windows* [23], no desarrolladas ni mantenidas por los desarrolladores de la aplicación original, y por tanto es una opción viable en estos sistemas. Sin embargo, estas versiones no oficiales van a la zaga de la original en su versión, y presentan poca información y documentación para su uso.

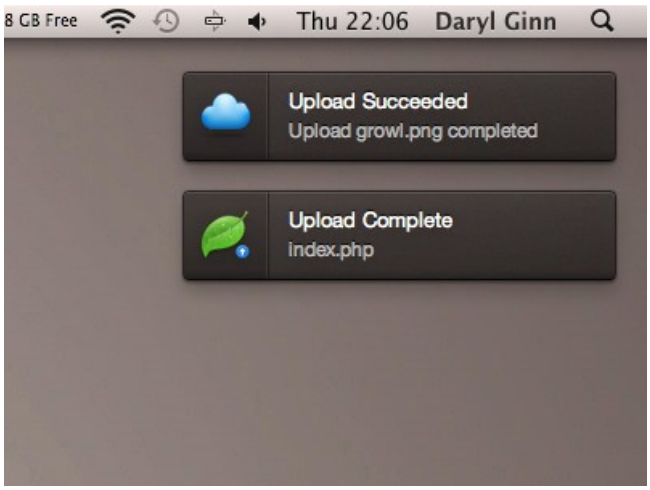

Ilustración 4: Ejemplo de estilo en Growl [24]

D-Bus y Desktop Notifications Specification

D-Bus es un sistema de comunicación interproceso basado en un bus de mensajes [25], que forma parte del proyecto de interoperabilidad y tecnologías base compartidas para entornos de escritorio en Linux y otros sistemas UNIX *freedesktop.org* [26].

Mediante la librería *libdbus* (u otra implementación alternativa, *libdbus* es la implementación de referencia desarrollada por el propio *freedesktop.org* pero existen otras adicionales) dos procesos pueden conectar entre sí y enviarse mensajes a través de un demonio que actúa a modo de bus de mensajes (*dbus-daemon*, implementado sobre *libdbus*), y que dirige a cada proceso de destino los mensajes correspondientes.

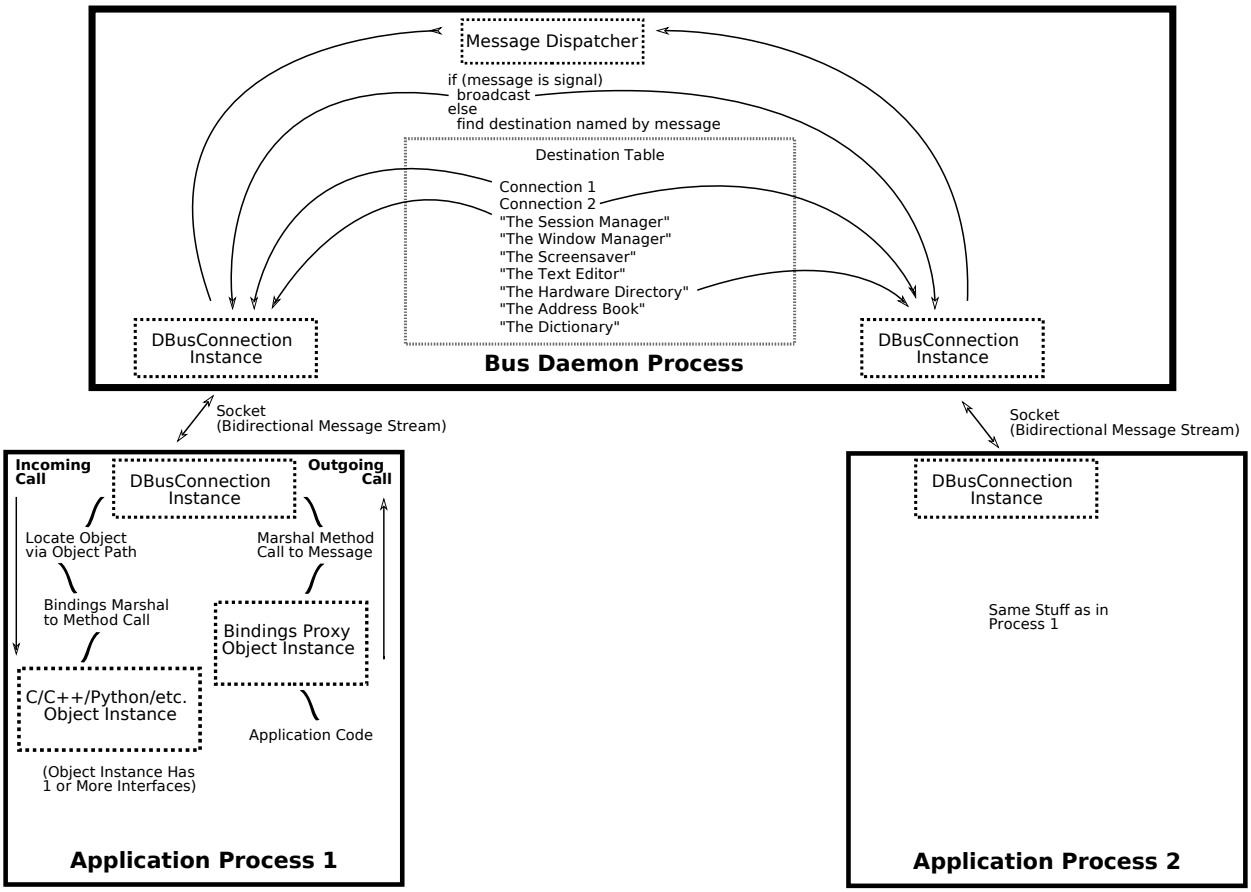

Ilustración 5: Diagrama de funcionamiento de D-Bus [25]

D-Bus tiene una extensiva adopción entre las distintas distribuciones GNU/Linux y existe incluso una versión del mismo para Windows.

Haciendo uso de *D-Bus* como interfaz de comunicación, la Especificación de Notificaciones de Escritorio (*Desktop Notifications Specification*)[27] es un estándar propuesto por el *GNOME*

Project para un servicio de notificaciones de escritorio, mediante el cuál las aplicaciones puedan generar ventanas emergentes pasivas con las que presentar notificaciones al usuario.

Esta especificación define un protocolo de comunicación a través de *D-Bus* entre las aplicaciones que deseen generar las notificaciones, y un servidor de notificaciones que recibe los avisos de estas aplicaciones y proporciona el servicio generando las ventanas emergentes pasivas correspondientes.

La mayoría de distribuciones de GNU/Linux incorporan un servidor de notificaciones basado en esta especificación que es utilizado por el propio sistema y muchas aplicaciones para mostrar notificaciones no intrusivas al usuario, por lo que hacer uso de este mecanismo tiene la ventaja de que las notificaciones que recibe el usuario están homogéneamente integradas con el resto del sistema.

libnotify

Libnotify es una librería mediante la cual se pueden enviar notificaciones de escritorio a un demonio de notificación [28]. Para ello, esta librería implementa la Especificación de Notificaciones de Escritorio del *GNOME Project* ofreciendo una sencilla API que utilizar en lugar de acceder al servidor de notificaciones directamente a través de una implementación de *D-Bus* como *libdbus* (que ofrece una API de muy bajo nivel). Es desarrollada y mantenida por el propio *GNOME Project*.

Libnotify soporta aplicaciones basadas en *GTK+* y *Qt* (lo que supone una dependencia extra aparte de la propia *libnotify* en caso de que no se requieran otras funcionalidades de alguna de las anteriores) y es independiente del escritorio utilizado gracias al uso del protocolo de notificaciones de *GNOME*.

Para que un programa que utilice *libnotify* muestre notificaciones al usuario, es un requisito que el sistema en el que se ejecuta disponga de un servidor de notificaciones compatible con la Especificación de Notificaciones de Escritorio. La mayoría de distribuciones GNU/Linux incorporan uno por defecto.

Notificaciones de navegador (Google Chrome)

Google Chrome, mediante la API para notificaciones *chrome.notifications*, ofrece la posibilidad de mostrar también notificaciones emergentes no intrusivas en el escritorio [29].

Esta API está disponible para ser usada desde *Chrome Apps*, aplicaciones multiplataforma que se ejecutan utilizando *Chrome* a modo de *runtime* y pueden por tanto ser ejecutadas en cualquier sistema compatible con el navegador (y que lo tenga instalado), incluyendo Linux, Windows, OS X, Android o iOS [30].

Las *Chrome Apps* se desarrollan utilizando tecnologías web (HTML, CSS, javascript…), pueden distribuirse a través de la tienda digital *Chrome Web Store*, y pueden lanzarse una vez instaladas desde el propio navegador o un lanzador de *Chrome Apps* que puede instalarse por separado. La dependencia del navegador para su despliegue y ejecución es su mayor contra.

La API *chrome.notifications* ofrece un conjunto de funcionalidades equiparable al de otras librerías de notificaciones de escritorio como *libnotify* o *Growl* en cuanto a las posibilidades de notificación, y ofrecen posibilidades adicionales en la forma de integración con otros servicios de Google como *Google Cloud Messaging* (del que se habla en el apartado 2.3.3).

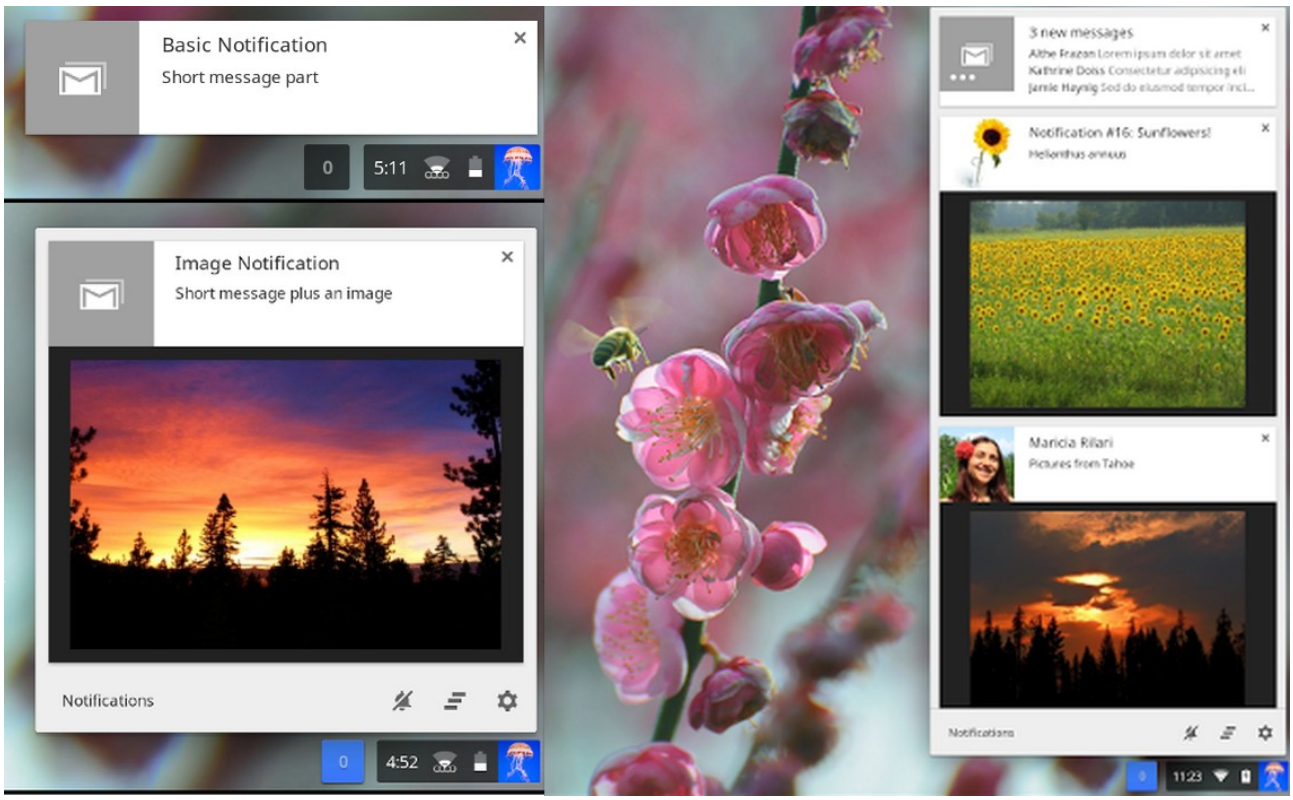

Ilustración 6: Ejemplos de notificaciones con Google Chrome [31]

2.3.3 Transmisión remota de eventos

Sockets POSIX (Berkeley sockets)

Al igual que en el caso de la E/S de ficheros, GNU/Linux proporciona una implementación de la interfaz estándar POSIX para comunicaciones de red mediante *sockets* como parte de sus librerías de sistema [18].

El modelo que define POSIX (también conocido como *Berkeley sockets*) utiliza el concepto de *socket* como una representación de una conexión entre dos sistemas como un cuarteto único compuesto por las dos direcciones IP y los dos puertos de los dos sistemas conectados, asignando a cada *socket* un descriptor de fichero para que el proceso que lo utiliza pueda acceder a él, y leer y escribir (recibir y enviar) datos en él.

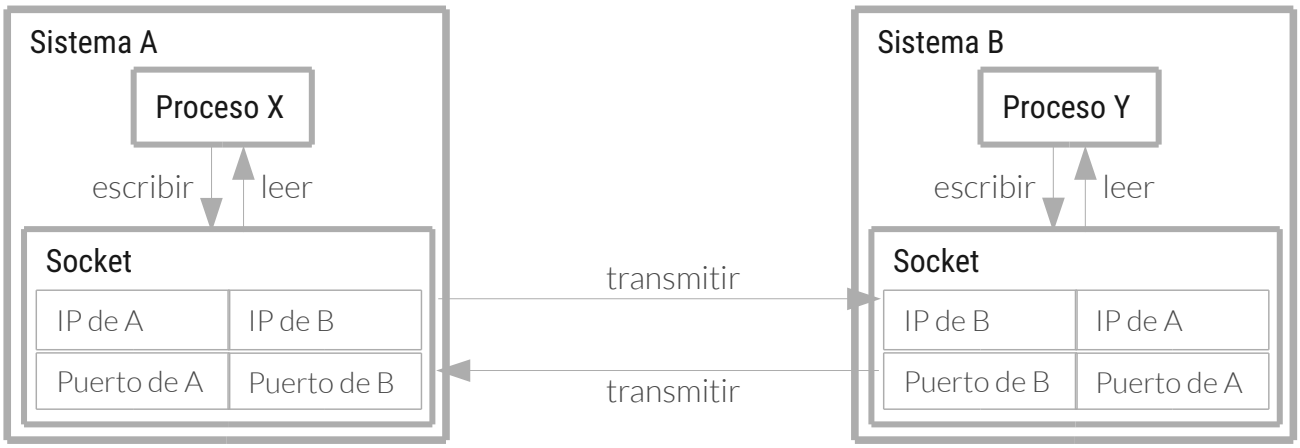

Ilustración 7: Modelo de sockets POSIX

El modelo de *sockets* POSIX permite la comunicación remota mediante protocolo orientado a conexión (TCP), no orientado a conexión (UDP), o incluso un nivel por debajo de éstos permitiendo al usuario definir su propio protocolo de conexión sin tener que hacerlo sobre uno de los anteriores.

Al ser un estándar POSIX, es portable entre sistemas GNU/Linux o UNIX, y en el caso del estándar de *sockets* POSIX incluso Windows dispone de una implementación en sus librerías de sistema [32].

Por supuesto al estar diseñado como una API de sistema para comunicaciones de red, se trata de una API de bajo nivel, que requiere definir y controlar de forma manual la práctica totalidad de los mecanismos de comunicación que desean implementarse más allá del comportamiento de los protocolos de conexión utilizados, por lo que la utilización de esta API para sistemas de comunicación complejos puede resultar un proceso costoso.

Librerías de funciones de red (Boost.Asio, RakNet, ENet)

Existen una gran cantidad de librerías que proporcionan APIs de más alto nivel que los *sockets* POSIX, por regla general implementadas sobre estos, con objeto de proporcionar métodos de conexión menos costosos de utilizar y/o con funcionalidades adicionales así como protocolos de comunicación de más alto nivel que TCP/UDP.

Un ejemplo de este tipo de librerías es *Boost.Asio* (*Boost* Asynchronous I/O) [33], que ofrece una API multiplataforma para E/S de red asíncrona de bajo nivel para C++. Es parte del paquete de librerías de C++ *Boost* [34], de código abierto y licencia gratuita. *Boost.Asio* no es en realidad una API de mucho más alto nivel que los *sockets* Berkeley, pero proporciona una interfaz con mejor integración con el lenguaje C++, y algunas otras abstracciones adicionales que pueden resultar provechosas.

Otro ejemplo es la librería *ENet* [35], cuyo propósito es proporcionar comunicaciones en red a través de un protocolo construido como una capa sobre UDP, agregando algunas funciones de TCP de forma opcional como la entrega garantizada y ordenada de los datos, pero de una forma más orientada a maximizar el rendimiento y velocidad que TCP. *ENet* es también código abierto de licencia gratuita.

Un caso de librería de más alto nivel que las anteriores es *RakNet* [36], una librería diseñada para juegos multi-jugador en línea, pero que ofrece una gran cantidad de funcionalidades que pueden ser aprovechadas para otros casos de uso, como conexiones seguras (SHA1, AES128, SYN Cookies y RSA) o *NAT traversal*. Es multiplataforma y puede ser utilizada en C++ y C#. Originalmente era una librería de licencia comercial, pero tras la compra de la misma por parte de Oculus, ha sido publicada como software libre y con licencia gratuita.

Todas estas librerías ofrecen ciertas ventajas sobre el uso de *sockets* POSIX a la hora de emplear comunicaciones de red, son multiplataforma, código abierto y de licencia gratuita.

Sin embargo, utilizar cualquiera de ellas supone incluir una dependencia adicional para el proyecto, y cada una presenta una API y unos protocolos diferentes a los estándares POSIX omnipresentes en la mayoría de los sistemas operativos de hoy.

Servicios *push* en la nube (Google Cloud Messaging)

Con el auge de las *apps* para dispositivos móviles inteligentes y el crecimiento de la oferta de servicios en la nube, algunos de los proveedores de este tipo de servicios ahora proporcionan servicios de notificaciones *push* para aplicaciones:

El servidor de notificaciones se registra con el servicio *push* en la nube, y los usuarios de las aplicaciones cliente, conectan también con el servicio a través de un canal de comunicación propio. De esa forma, el servidor puede enviar notificaciones al servicio *push* en la nube, y éste retransmite la notificación a todas las aplicaciones cliente conectadas.

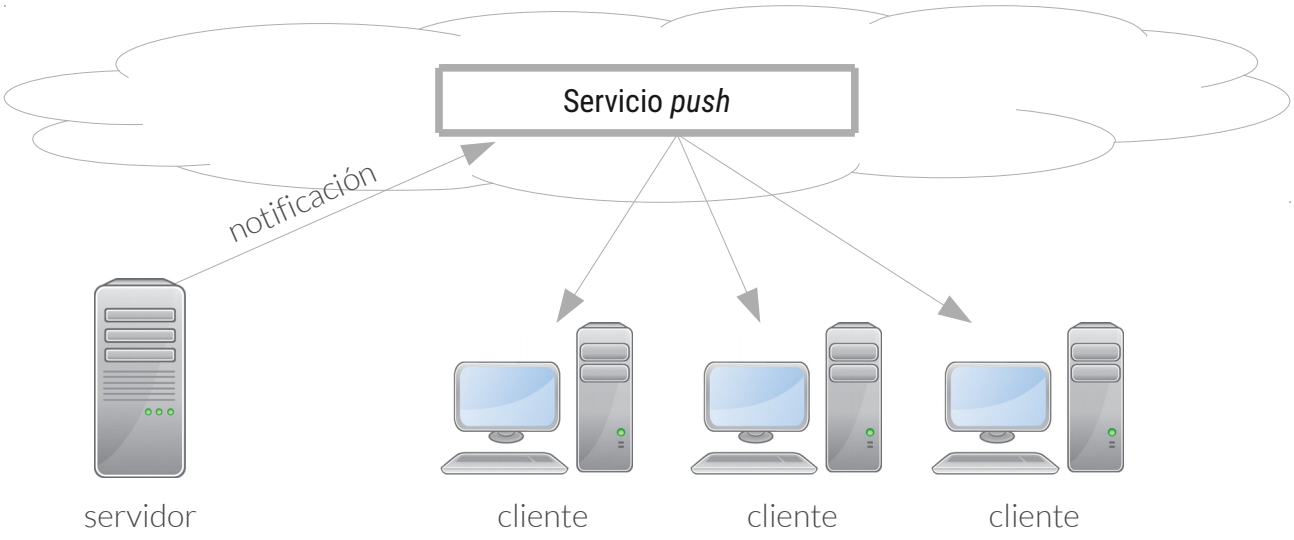

Ilustración 8: Modelo de servicio push en la nube

Este tipo de servicios suele ofrecer funcionalidades de alto nivel como conexiones seguras, sistemas de autenticación de usuarios y cuentas, notificaciones diferidas (cuando un cliente no está conectado en el momento de ser emitida una notificación, aún puede recibirla más adelante en el momento de conectar) y APIs de alto nivel, así como interacción con otros servicios y funcionalidades. Por otro lado, suelen requerir cuentas de usuario del servicio para utilizar las aplicaciones clientes, cuentas especiales de desarrollador para registrar las aplicaciones servidoras, y en ocasiones imponen limitaciones en las plataformas en las que las aplicaciones cliente pueden recibir las notificaciones.

Por ejemplo, *Google Cloud Messaging* [37] requiere registrar tanto la aplicación servidora como la cliente, requiere una cuenta de *Google Developer*, y está diseñado para aplicaciones cliente en plataformas Android, iOS, y aplicaciones Chrome (*Chrome Apps* [30]), dejando de lado aplicaciones para de escritorio PC salvo a través del navegador Chrome.

3 Desarrollo de la solución

3.1 Metodología

Para el desarrollo de este proyecto se ha decidido utilizar una metodología secuencial correspondiente al modelo conocido como **desarrollo en cascada**, enfoque metodológico que ordena las diferentes etapas del proceso de desarrollo como una serie de fases consecutivas cada una de las cuales debe ser finalizada por completo antes de avanzar a la siguiente.

Las etapas seguidas para el desarrollo en cascada del proyecto son:

1. **Análisis:** evaluación y especificación de los requisitos funcionales y no funcionales del software a desarrollar así como de los posibles casos de uso del mismo, definiendo de forma concreta las objetivos a cubrir durante el desarrollo.
2. **Diseño:** definición y organización de los componentes del sistema, su arquitectura y las interacciones entre los mismos, describiendo la estructura relacional global del sistema.
3. **Implementación:** desarrollo del código necesario para crear el sistema diseñado en la etapa anterior como un software funcional que cumpla los requisitos y objetivos planteados en la primera etapa.
4. **Pruebas:** ensamblado y puesta en uso de prueba del software implementado para revisar que toda la funcionalidad implementada en el software funciona sin errores.

Otras etapas que típicamente se siguen en un modelo de desarrollo en cascada para muchos proyectos, como *Implantación*, *Soporte* y *Mantenimiento*, se han descartado como parte de este proyecto debido a la naturaleza del mismo (el objetivo final es el desarrollo de un *prototipo funcional* a modo de prueba de concepto, no un sistema para ser distribuido o implantado de forma definitiva a un usuario final).

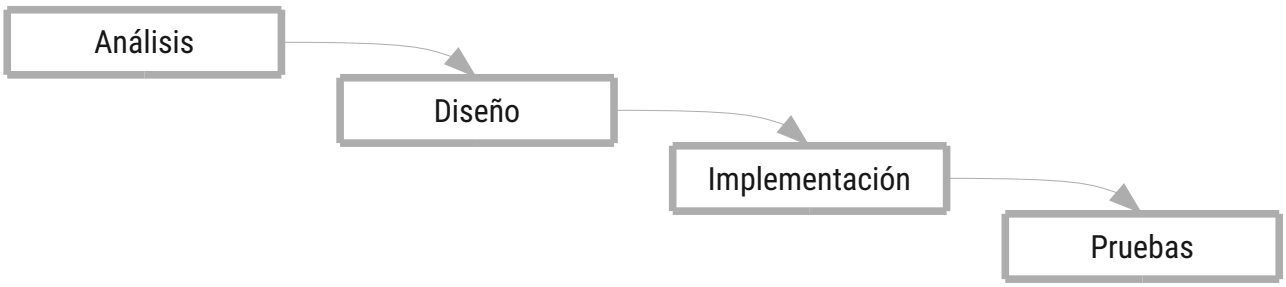

Ilustración 9: Proceso de desarrollo en cascada

Existen modelos de desarrollo alternativos a este, como por ejemplo el *modelo de prototipos* (basado en el desarrollo de sucesivos prototipos incompletos del producto final al que se le van añadiendo funcionalidades y módulos en sucesivas etapas), el *modelo en espiral* (que combina aspectos del modelo de prototipos y el modelo en cascada) y otras *metodologías ágiles* (basadas también en procesos de desarrollo iterativos con una aproximación más ligera y basada en *feedback* para las etapas de desarrollo).

Todos estos modelos y metodologías presentan soluciones a las limitaciones asociadas al proceso de desarrollo secuencial en cascada:

- Falta de flexibilidad para adaptarse a requisitos cambiantes, por ejemplo cuando el cliente no tiene claras sus necesidades desde el comienzo. Los cambios en los requisitos del proyecto suponen rediseño, volver a implementar y a hacer pruebas, aumentando los costes de desarrollo.
- Falta de flexibilidad para adaptarse a dificultades futuras que puedan aparecer durante una etapa de desarrollo que no han sido previstas en las anteriores. Por ejemplo, ciertas decisiones de diseño pueden conducir a dificultades que se descubren durante la implementación, que podrían corregirse más fácilmente alterando el diseño para adaptarlas a ellas. Los modelos iterativos permiten anticipar estas dificultades y corregirlas con anterioridad.
- Una etapa determinada del proyecto no puede llevarse a cabo hasta que todas las anteriores han sido finalizadas, lo que supone que durante el desarrollo de cada etapa no se dispone de ningún tipo de información que pudiera obtenerse de las etapas subsiguientes, y específicamente, no se dispone de código funcional hasta las últimas etapas del proyecto.

Si bien es cierto que estas limitaciones están perfectamente fundamentadas y presentan argumentos sólidos para utilizar modelos de ciclo de vida del proyecto más iterativos que el modelo en cascada, existen también una serie de condicionantes asociados a la naturaleza de este proyecto que han decantado la elección en su favor:

- Los requisitos del proyecto pueden ser definidos de forma clara desde el principio, y ser mantenidos de forma inmutable a lo largo del ciclo de vida del proyecto.
- Tratándose de un modelo lineal, es sencillo y rápido de implementar.
- Una etapa finalizada proporciona una base estable sobre la que cimentar la siguiente, y la planificación resultante es clara, estructurada y más fácil de predecir.

3.2 Marco regulador

Este proyecto presenta un sistema cuyo fin último es la transmisión y presentación visual de datos originados a partir del uso y funcionamiento del mismo u otros equipos informáticos que aquel en que son presentados, por lo que es necesaria la consideración de la legislación vigente relacionada, así como con la utilización de propiedad intelectual de terceras partes.

Protección de Datos de Carácter Personal

Concretamente, la ***Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal*** [38] regula el uso de <<[...] los datos de carácter personal registrados en soporte físico, que los haga susceptibles de tratamiento, y a toda modalidad de uso posterior de estos datos por los sectores público y privado>> y tiene por objeto <<[...] garantizar y proteger, en lo que concierne al tratamiento de los datos personales, las libertades públicas y los derechos fundamentales de las personas físicas, y especialmente de su honor e intimidad personal y familiar>>.

En relación a la misma, se considera que:

- Según el *Artículo 3 apartado a* se definen "Datos de carácter personal" como <<[...] cualquier información concerniente a personas físicas identificadas o identificables>>. El sistema desarrollado **no** proporciona de forma directa ningún medio que permita asociar los datos transmitidos a una identidad personal concreta, y los ficheros de registro de sistema para los que se propone su uso, nuevamente contienen información de funcionamiento del sistema informático al que pertenecen, no vinculables a una persona física identificada o identificable a partir de la información proporcionada por el sistema.
- Según el *Artículo 2 apartado 2a* dicha Ley no será de aplicación a <<[...] los ficheros mantenidos por personas físicas en el ejercicio de actividades exclusivamente personales o domésticas>>. El objetivo del presente proyecto es desarrollar un *prototipo funcional* de un sistema no destinado a su implantación fuera del ámbito personal .

Por lo tanto, para el uso pretendido y propuesto de este sistema la Ley Orgánica mencionada **no es de aplicación**. Sin embargo, es importante considerar que ciertos usos más allá de estos que puedan involucrar el seguimiento de ficheros de registro con contenidos de datos personales no anónimos en ámbitos de múltiples usuarios (como su implantación en los sistemas informáticos de la universidad o de una empresa) pueden requerir medidas

adicionales incluyendo (pero no limitándose a) la solicitud expresa y por escrito del consentimiento de los usuarios afectados o garantizar la seguridad de los datos para asegurar la conformidad con esta ley.

Protección de la Propiedad Intelectual

En cuanto a la legislación concerniente a la utilización en el presente proyecto de Propiedad Intelectual perteneciente a terceras partes, se encuentra en el ***Real Decreto Legislativo 1/1996, de 12 de abril*** con el texto refundido de la ***Ley de Propiedad Intelectual*** [39].

Como parte de este proyecto se hace uso a modo de componentes externos de las librerías *GTK+* y *libnotify*.

GTK+ es software libre y parte del Proyecto GNU (GNU Project), y permite la utilización por parte de cualquier desarrollador (incluyendo aquellos desarrollando software propietario) sin ninguna tasa de licencia o derechos [40] a través de la licencia **GNU Lesser General Public License** (**LGPL**) en su versión 2.1 [41]. Esta licencia requiere hacer público el código correspondiente a cualquier modificación realizada sobre la librería utilizada; en este caso se utiliza el código tal cuál sin modificación.

Libnotify es software libre y parte del Proyecto GNOME (GNOME Project), y sus términos de utilización [42] son también expuestos mediante la misma versión (2.1) de la GNU Lesser General Pulic License (LGPL) mencionada anteriormente; también se utiliza el código de la librería tal y como se proporciona sin ninguna modificación.

Publicación y licencia de uso

El presente proyecto está publicado como software libre en Github (en la dirección URL: https://github.com/guillermofarina/lognotify). Se han considerado las siguientes licencias como posibilidades para la publicación del mismo:

- **GNU General Public License version 3** (**GPL v3**) [43]: esta licencia para software libre proporciona a cualquier desarrollador el derecho de usar el código licenciado de forma libre, incluyendo la capacidad de usar, estudiar, compartir y modificar el software licenciado, siempre y cuando el nuevo código derivado o que utilice el código licenciado con GPL sea publicado a su vez como software libre bajo la misma licencia GPL.
- **GNU Lesser General Public License version 3** (**LGPL v3**) [41]: es una variante más permisiva de la anterior según la cuál es posible utilizar el software licenciado LGPL en software privativo siempre y cuando el mismo sea utilizado sin modificación en el

software que lo usa. Trabajos derivados que impliquen la alteración del código proporcionado requieren hacer públicas dichas modificaciones para cumplir con la licencia.

- **MIT License** (**MIT**) [44]: la licencia MIT es la más permisiva entre las contempladas, proporcionando a cualquier persona que obtenga una copia del software el derecho sin coste alguno de utilizar sin restricción el mismo, incluyendo copia, modificación, fusión, publicación, distribución, y venta de copias.

	GPL v3	LGPL v3	MIT
Requiere[1] cesión gratuita de derechos de uso del SW licenciado			
Permite la utilización del SW sin restricciones			
Incluyendo el uso comercial del mismo			
Permite el uso de otras licencias en el SW derivado (no es *copyleft*)			
Permite la publicación de software derivado no libre			
Permite modificar el SW sin publicar los cambios introducidos			

[1]*Aunque no requiera la cesión gratuita de los derechos, todos la permiten*
Permitido / requerido (según el punto) por la licencia en cuestión
No permitido / requerido (según el punto) por la licencia en cuestión

Tabla 2: Tabla comparativa (licencias contempladas para la publicación del proyecto)

Tras una valoración de las licencias consideradas, se considera que para el presente proyecto la licencia LGPL no es la más apropiada, puesto que está más enfocada a librerías o *middleware*, y dado que en cualquier caso la cesión de derechos de utilización del software se proporcionará de forma gratuita y sin coste, el criterio diferenciador entre las dos restantes es la permisión del uso privativo sin restricciones del SW proporcionado (MIT) o una licencia *copyleft* que obligue a publicar como software libre cualquier trabajo derivado del mismo o modificación realizada (GPL).

Se considera que la última opción es más favorable para un desarrollo futuro del software proporcionado, por lo que la licencia escogida para el proyecto es **GNU General Public License** (**GPL**) **version 3**.

3.3 Análisis

El objetivo de la etapa de análisis es definir y documentar de forma concreta y específica los requerimientos del proyecto, definiendo el problema a través de una especificación de los requisitos y escenarios de utilización para el sistema a desarrollar.

3.3.1 Especificación de requisitos

La especificación de requisitos del sistema a desarrollar define las condiciones para que éste cumpla con los objetivos propuestos y los criterios de acuerdo a los cuales éste debe de ser desarrollado.

Cada requisito describe una característica o cualidad específica que el sistema desarrollado debe poseer.

Los requisitos especificados pueden ser de dos tipos, *requisitos funcionales* o *requisitos no funcionales*:

- Los requisitos funcionales especifican servicios que debe proporcionar el sistema, así como el comportamiento del sistema ante entradas o situaciones particulares. Describen, en suma, lo que debe hacer o permitir hacer el sistema desarrollado.
- Los requisitos no funcionales especifican restricciones aplicables a los servicios o funciones ofrecidos por el sistema o al proceso de desarrollo, incluyendo cualidades relacionadas con el rendimiento, la usabilidad, la adecuación a estándares, u otras cuestiones similares.

La definición de cada requisito incluye:

- **Identificador:** Código alfanumérico que identifica de forma unívoca el requisito en cuestión. El formato de dicho identificador es RF-*n* para requisitos funcionales, RNF-*n* para requisitos no funcionales, donde *n* es un número entero igual o mayor que 1.
- **Descripción:** Declaración textual concisa del requerimiento o restricción que supone el requisito.
- **Motivación:** Descripción textual concisa del motivo para la inclusión del requisito.
- **Aprobación:** Descripción textual concisa y no ambigua del criterio utilizado para determinar si el sistema cumple el requisito en cuestión o no.
- **Dependencias:** Enumeración de otros requisitos que dependen de éste, si existen.

- **Importancia:** Declaración del nivel de importancia del cumplimiento del requisito, en relación al impacto de su no cumplimiento para el proyecto. Según su importancia un requisito puede ser *Obligatorio* (el cumplimiento del requisito es condición necesaria para la finalización con éxito del proyecto) u *Opcional* (el cumplimiento del requisito no es condición necesaria para la finalización del proyecto).
- **Prioridad:** Asignación de la prioridad relativa de un requisito (a mayor prioridad, antes debe ser implementado). La prioridad se asigna en función de su importancia relativa y de los requisitos dependientes del mismo, y puede ser *Alta*, *Moderada* o *Baja*.

Requisitos funcionales

Identificador	Descripción		
RF-1	El sistema debe permitir monitorizar múltiples ficheros de registro específicos por equipo		
Motivación:	El SO y otros programas instalados utilizan múltiples ficheros de registro para distintas funciones. Varios de estos pueden requerir monitorización, varios no, y pueden existir motivos por los que se quiera evitar mostrar algunos de ellos		
Aprobación:	El sistema da la posibilidad de especificar uno o más ficheros para monitorizar, sin limitar la cantidad de estos que el usuario puede especificar. Estos y sólo estos serán monitorizados		
Dependencias:	RF-2, RF-6, RNF-3		
Importancia:	Obligatorio	Prioridad:	Alta

Tabla 3: RF-1

Identificador	Descripción		
RF-2	El sistema debe permitir al SO del equipo monitorizado realizar cualesquiera tareas de mantenimiento con sus ficheros de registro sin ver afectada su funcionalidad		
Motivación:	El SO realizará de manera rutinaria y periódica operaciones de manenimiento incluyendo borrado, renombrado y rotación de ficheros de registro. El sistema desarrollado debe por tanto soportar esto		
Aprobación:	Tras una rotación de ficheros de registro que afecte a uno o más ficheros monitorizados, el sistema continúa notificando eventos sobre los ficheros de nombre especificado previamente por el usuario, sin requerir ningún tipo de interacción adicional por parte del mismo		
Dependencias:			
Importancia:	Obligatorio	Prioridad:	Moderada

Tabla 4: RF-2

Identificador	Descripción		
RF-3	Un usuario debe poder llevar a cabo el proceso de montorización desde un equipo informático distinto al monitorizado		
Motivación:	El objetivo del sistema es proporcionar un medio de monitorización que pueda ser utilizado mientras se desempeñan otras actividades con la menor intrusividad posible en estas. Es probable que dichas actividades no se lleven a cabo en el equipo monitorizado		
Aprobación:	Puede monitorizarse un equipo desde otro diferente y el usuario recibe idéntica salida por parte del sistema que si lo hiciera en el equipo monitorizado		
Dependencias:	RF-4, RF-5, RNF-4, RNF-5		
Importancia:	Obligatorio	Prioridad:	Alta

Tabla 5: RF-3

Identificador	Descripción		
RF-4	Múltiples usuarios desde múltiples ubicaciones físicas deben poder monitorizar un mismo equipo de forma simultánea		
Motivación:	Es probable que se desee disponer de más de una persona responsable de la monitorización o que compartan el acceso al sistema; la presencia de uno no debe prevenir la de otros		
Aprobación:	Dos o más usuarios diferentes pueden monitorizar un mismo equipo y reciben idéntica salida por parte del sistema al hacerlo		
Dependencias:			
Importancia:	Obligatorio	Prioridad:	Moderada

Tabla 6: RF-4

Identificador	Descripción		
RF-5	Un usuario debe poder monitorizar múltiples equipos desde su puesto de trabajo		
Motivación:	Con frecuencia el objeto del proceso de monitorización será un conjunto de ordenadores en lugar de uno sólo. No debería requerirse más de una persona encargada para ello		
Aprobación:	Un usuario monitoriza desde un único equipo dos o más equipos diferentes, y recibe una salida idéntica a la adición de las salidas de monitorizar cada uno de estos equipos por separado		
Dependencias:			
Importancia:	Obligatorio	Prioridad:	Moderada

Tabla 7: RF-5

Identificador	Descripción		
RF-6	El sistema debe mostrar a cada usuario notificaciones de todas las entradas añadidas a ficheros de registro monitorizados		
Motivación:	Para que el proceso de monitorización pueda llevarse a cabo, el encargado del mismo debe recibir notificación de los eventos relevantes, en este caso, todas las entradas que el SO agrega a los ficheros de registro monitorizados		
Aprobación:	Siempre que una nueva entrada es añadida a un fichero monitorizado, un usuario monitorizando ese sistema recibe una notificación mostrando la entrada añadida al fichero modificado		
Dependencias:	RF-7, RF-8, RNF-6		
Importancia:	Obligatorio	Prioridad:	Alta

Tabla 8: RF-6

Identificador	Descripción		
RF-7	Un usuario debe poder acotar (reducir) el número de notificaciones que le son mostradas a su discreción		
Motivación:	Cuantas menos notificaciones muestre el sistema, menos intrusivo resulta para el usuario en el desempeño de otras actividades. Un usuario puede disponer con antelación de criterios que determinen ciertos eventos como irrelevantes, y omitir estos proporciona una mejor experiencia de uso		
Aprobación:	El usuario puede configurar el sistema de forma que éste omita de forma selectiva la notificación de ciertos eventos en base al equipo del que proceden, el fichero de registro implicado o el contenido de la entrada añadida		
Dependencias:			
Importancia:	Opcional	Prioridad:	Baja

Tabla 9: RF-7

Identificador	Descripción		
RF-8	Un usuario debe poder establecer sus preferencias en la forma de visualización de las notificaciones		
Motivación:	Cada usuario puede tener diferentes preferencias sobre la información mostrada por los mensajes de notificación y cómo se muestra ésta		
Aprobación:	El sistema permite al usuario definir al menos el tiempo que permanecen en pantalla las notificaciones, y preferiblemente otras opciones de visualización (como mostrar o no ciertos campos, o dónde)		
Dependencias:			
Importancia:	Opcional	Prioridad:	Baja

Tabla 10: RF-8

Identificador	Descripción		
RF-9	Un usuario debe poder revisar a posteriori la totalidad de las notificaciones recibidas		
Motivación:	Dado que las notificaciones pueden ser ignoradas, es conveniente que un usuario pueda ver después del hecho el conjunto de notificaciones recibidas y cuándo han ocurrido. Además esto permite obtener una visión de conjunto de los hechos transcurridos. Esto permite una funcionalidad similar a la de revisar los propios ficheros de registro monitorizados, sin tener que acceder a través del equipo en que se encuentran		
Aprobación:	El sistema registra en un fichero de texto todos los eventos recibidos por el usuario por estricto orden de llegada y acompañados de la fecha y hora del mismo, en un formato legible por una persona.		
Dependencias:			
Importancia:	Opcional	Prioridad:	Baja

Tabla 11: RF-9

Requisitos no funcionales

Identificador	Descripción		
RNF-1	La puesta en marcha del sistema debe ser un proceso compatible con automatización completa		
Motivación:	Los procesos de monitorización suelen ser tareas recurrentes que se realizan de forma reiterada y/o rutinaria. Es deseable que el sistema pueda (si se desea) ser lanzado de forma automática en el arranque de los equipos informáticos en que se encuentra sin necesidad de intervención humana		
Aprobación:	Todas las entradas requeridas para la ejecución de los entregables ejecutables del proyecto pueden ser especificadas como parámetros por línea de comandos y/o leídos de ficheros en disco, y no requieren de ninguna interacción adicional con usuarios humanos para su puesta en marcha		
Dependencias:			
Importancia:	Opcional	Prioridad:	Baja

Tabla 12: RNF-1

Identificador	Descripción		
RNF-2	El sistema debe poder ofrecer su funcionalidad completa funcionando en segundo plano		
Motivación:	Como parte del objetivo de resultar lo menos intrusivo posible, el sistema debe poder ofrecer su funcionalidad de la forma más transparente posible para el usuario, dando las mínimas muestras posibles de su presencia en el sistema		
Aprobación:	Todos los entregables ejecutables del proyecto pueden ejecutarse como procesos demonio		
Dependencias:			
Importancia:	Opcional	Prioridad:	Baja

Tabla 13: RNF-2

Identificador	Descripción		
RNF-3	Rutas válidas de ficheros a monitorizar no deben limitarse a la especificación de la FSH		
Motivación:	No todos los sistemas GNU/Linux siguen la especificación de la FSH, y sistemas individuales pueden ser configurados de diferentes formas		
Aprobación:	El sistema permite especificar una ruta para el directorio donde el sistema ubica los ficheros de registro (por omisión, usa la especificación FSH y es /var/log) incluyendo el directorio raíz (/), y permite especificar rutas relativas a ésta para cada fichero, incluyendo subdirectorios		
Dependencias:			
Importancia:	Obligatorio	Prioridad:	Moderada

Tabla 14: RNF-3

Identificador	Descripción		
RNF-4	El protocolo de transmisión utilizado para la monitorización a distancia debe ofrecer garantía de entrega y corrección en el envío de datos		
Motivación:	Para que el sistema de monitorización sea fiable, no deben poder perderse notificaciones de eventos provenientes de equipos monitorizados		
Aprobación:	El sistema utiliza un protocolo de conexión que proporcione garantía de entrega y corrección de los datos		
Dependencias:			
Importancia:	Obligatorio	Prioridad:	Moderada

Tabla 15: RNF-4

Identificador	Descripción		
RNF-5	El mecanismo de conexión a distancia utilizado debe ser compatible con acceso a través de Internet además de LAN		
Motivación:	Permitir el acceso a través de Internet a los equipos monitorizados otorga una mayor flexibilidad al usuario en cuanto a posibilidades a la hora de realizar la tarea de monitorización		
Aprobación:	El sistema utiliza un medio de transmisión compatible con acceso a través de LAN e Internet		
Dependencias:			
Importancia:	Opcional	Prioridad:	Baja

Tabla 16: RNF-5

Identificador	Descripción		
RNF-6	Las notificaciones de aviso no deben resultar intrusivas		
Motivación:	El usuario debe poder realizar la tarea de monitorización de forma pasiva y mientras desempeña otras actividades. Es por tanto importante que el sistema de monitorización resulte lo menos intrusivo posible		
Aprobación:	Los mensajes de notificación ocupan una posición periférica en la pantalla (contiguas a un borde o esquina de la misma) y permanecen solo unos segundos en esta antes de desaparecer por si solas si el usuario las ignora		
Dependencias:			
Importancia:	Obligatorio	Prioridad:	Moderada

Tabla 17: RNF-6

3.3.2 Casos de uso

El análisis de los casos de uso del sistema proporciona una descripción de los escenarios de uso y actividades que pueden realizarse con el sistema desde una perspectiva de sus interacciones con entidades externas al propio sistema, como sus usuarios u otros sistemas externos que se comuniquen con éste.

Las entidades externas al sistema se describen como **Actores**, mientras que cada escenario o actividad posible conforma un **caso de uso**.

Actores

Los siguientes actores (entidades externas al sistema que interactúan con el mismo) pueden ser identificados:

- **Administrador:** un actor con el rol de Administrador instala y configura el servicio proporcionado por el sistema.
- **Usuario:** un actor con el rol de Usuario hace uso del servicio proporcionado por el sistema, y puede especificar las condiciones particulares de uso para su sesión.

Cabe reseñar que los actores definidos especifican ***roles*** dentro del sistema, y no han de coincidir necesariamente con personas físicas: una misma persona puede actuar en varios roles diferentes, y por tanto interactuar con el sistema mediante casos de uso asociados a diferentes actores, y varias personas diferentes pueden actuar con un mismo rol.

La distinción de los diferentes roles (y por tanto actores) es importante en cuanto a que no necesariamente todos los usuarios del sistema propuesto disponen de permiso o privilegios para llevar a cabo ambas funciones.

Diagrama de casos de uso

El siguiente diagrama representa las interacciones de los actores identificados previamente con el sistema en la forma de escenarios de utilización del sistema o actividades llevadas a cabo con el mismo, es decir, ***casos de uso***.

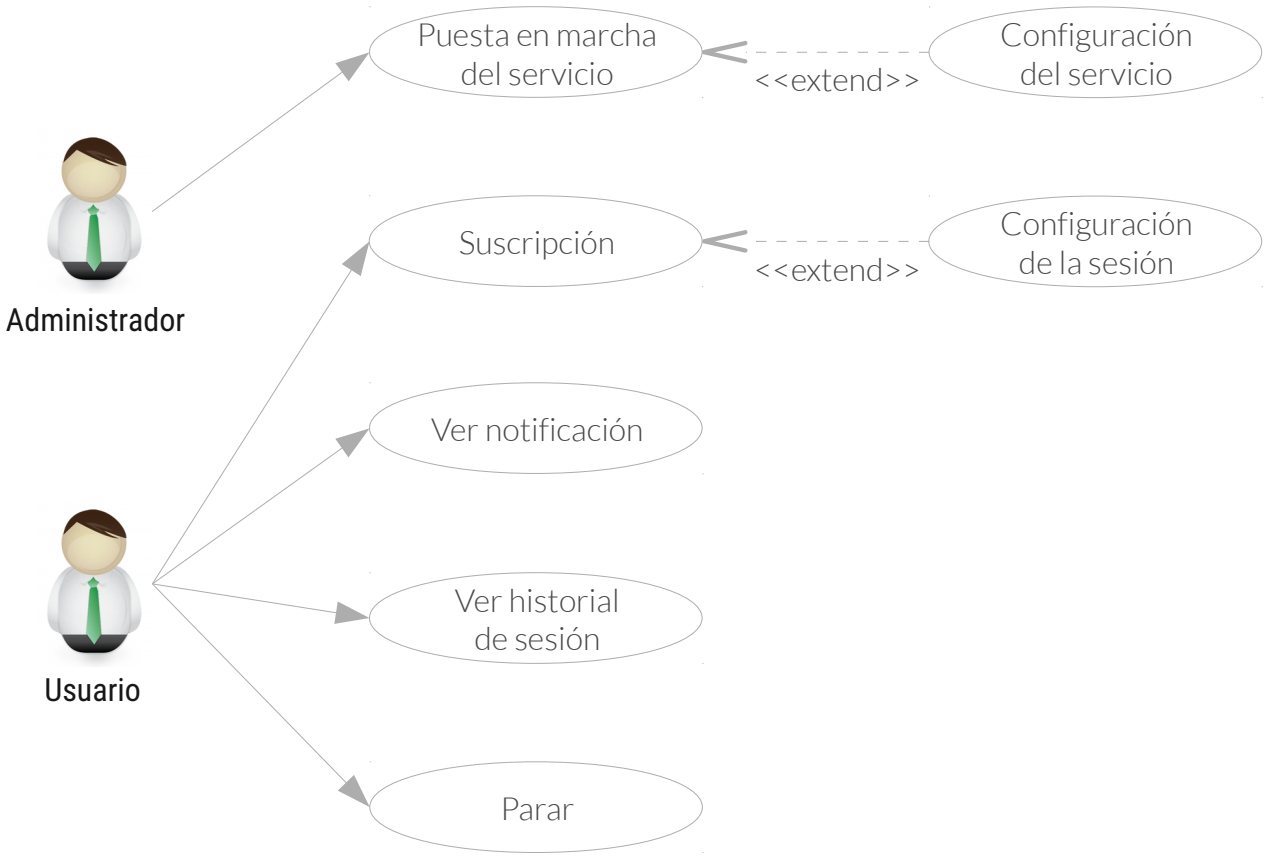

Ilustración 10: Diagrama de casos de uso

Descripción de casos de uso

Cada uno de los casos de uso mencionados anteriormente se describe con más detalle aportando información adicional que permita definir con más precisión cómo transcurre cada uno, o las condiciones en que lo hace.

Para cada caso de uso se define:

- **Identificador:** código alfanumérico que identifica de forma unívoca el caso de uso. El formato de dicho identificador es CU-*n*, donde *n* es un número entero mayor o igual que 1. En el caso de casos de uso de segundo orden (parte de otros casos de uso) el formato seguido es CU-*n.m*, donde *n* es el número correspondiente al caso de uso de primer orden relacionado, y *m* un número entero mayor o igual que 1.
- **Nombre:** título descriptivo textual que identifica al caso de uso indicando a qué hace referencia el mismo.
- **Actores:** actor o actores que toman parte en el caso de uso.
- **Propósito:** breve descripción textual de la intención del caso de uso.
- **Descripción:** breve descripción textual del caso de uso.
- **Pre-condiciones:** listado de condicionantes contextuales en el que el caso de uso es aplicable; si las pre-condiciones no se cumplen, el caso de uso no puede darse.
- **Curso del escenario básico:** descripción secuencial y enumerada del proceso a través del que transcurre el caso de uso en circunstancias normales. La secuencia se divide en dos columnas, una para **Acción de los actores** y otra para **Respuesta del sistema**, manteniendo para ambas una única secuencia numérica compartida que indica el orden en el que ocurre cada paso.
- **Curso de escenario/s alternativo/s:** *si existen* escenarios alternativos, estos se describen a continuación del escenario básico, siguiendo el mismo formato.
- **Post-condiciones:** listado de condicionantes contextuales que pasan a cumplirse una vez se ha desarrollado el caso de uso.

Identificador:	CU-1	
Nombre:	Puesta en marcha del servicio	
Actores:	Administrador	
Propósito:	Iniciar el servicio proporcionado por el sistema para que los usuarios puedan subscribirse	
Descripción:	El Administrador arranca manualmente el servicio para el equipo informático en que está instalado. Opcionalmente y como paso previo, puede establecer o modificar la configuración del servicio (CU-1.1)	
Pre-condiciones:	Ninguna	
Curso del escenario básico		
	Acción de los actores	Respuesta del sistema
1.	(opcional) <<punto de extensión>> CU-1.1	
2.	El Administrador introduce los parámetros del servicio (como el puerto de escucha)	
3.	El Administrador ejecuta la aplicación del servicio	
4.		El Sistema pone en marcha el servicio, pasando a admitir suscripciones
Curso de escenario alternativo: parámetros o configuración no válidos		
	Acción de los actores	Respuesta del sistema
1.	(opcional) <<punto de extensión>> CU-1.1	
2.	El Administrador introduce unos parámetros del servicio erróneos	
3.	El Administrador ejecuta la aplicación del servicio	
4.		El Sistema no pone en marcha el servicio, informando en su lugar al Administrador del error
Post-condiciones:	El equipo informático en cuestión ofrece servicio	

Tabla 18: CU-1 Puesta en marcha del servicio

Identificador:	CU-1.1
Nombre:	Configuración del servicio
Actores:	Administrador
Propósito:	Indicar al sistema los ficheros de registro que deben ser monitorizados y la ubicación de los mismos
Descripción:	El Administrador lista en un fichero de configuración los ficheros de registro que deben ser monitorizados en el equipo informático en cuestión
Pre-condiciones:	Ninguna

Curso del escenario básico	
Acción de los actores	Respuesta del sistema
1. El Administrador abre el fichero de configuración	
2. El Administrador escribe, modifica o mantiene como estaba la ruta del directorio de ficheros de registro (/var/log por defecto)	
3. El Administrador añade, elimina o modifica el nombre de un fichero a monitorizar	
4. El Administrador repite el paso 3 hasta estar conforme con la configuración establecida	
5. El Administrador guarda y cierra el fichero de configuración	
Post-condiciones: Ninguna	

Tabla 19: CU-1.1 Configuración del servicio

Identificador:	CU-2
Nombre:	Suscripción
Actores:	Usuario
Propósito:	Iniciar una sesión de monitorización para empezar a recibir notificaciones de eventos en su equipo provenientes de los equipos a los que se suscribe
Descripción:	El Usuario lanza la aplicación cliente en el equipo informático que está utilizando. Opcionalmente y como paso previo, puede establecer o modificar la configuración de la sesión (CU-2.1)
Pre-condiciones:	Los equipos informáticos a los que quiere suscribirse ofrecen servicio

Curso del escenario básico	
Acción de los actores	Respuesta del sistema
1. (opcional) <<punto de extensión>> CU-2.1	
2. El Usuario introduce los parámetros de ejecución que desee	
3. El Usuario ejecuta la aplicación cliente	
	4. El Sistema se suscribe a todos los servidores dados en la configuración
	5. El Sistema da inicio a la sesión

Curso de escenario alternativo: parámetros o configuración no válidos	
Acción de los actores	Respuesta del sistema
1. (opcional) <<punto de extensión>> CU-1.1	
2. El Usuario introduce unos parámetros de ejecución erróneos	
3. El Usuario ejecuta la aplicación cliente	
	4. El Sistema no se suscribe al servicio ni inicia la sesión, informando en su lugar al Usuario del error

Post-condiciones:	Sesión de monitorización iniciada

Tabla 20: CU-2 Suscripción

Identificador:	CU-2.1
Nombre:	Configuración de la sesión
Actores:	Usuario
Propósito:	Indicar al sistema los equipos informáticos a los que quiere suscribirse para recibir servicio, y qué notificaciones desea recibir
Descripción:	El Usuario lista en un fichero de configuración los servidores y configura el filtrado de notificaciones en otro
Pre-condiciones:	Ninguna

Curso del escenario básico	
Acción de los actores	Respuesta del sistema
1. El Usuario abre el fichero de servidores	
2. El Usuario añade, elimina o modifica la dirección de un servidor a la lista	
3. El Usuario repite el paso 3 hasta estar conforme con la lista resultante	
4. El Usuario guarda y cierra el fichero de servidores	
5. El Usuario abre el fichero de reglas de filtrado	
6. El Usuario añade, elimina o modica una regla de filtrado	
7. El Usuario repite el paso 6 hasta estar conforme con el conjunto de reglas	
8. El Usuario guarda y cierra el fichero de reglas de filtrado	

Post-condiciones:	Ninguna

Tabla 21: CU-2.1 Configuración de la sesión

Identificador:	CU-3
Nombre:	Ver notificación
Actores:	Usuario
Propósito:	Obtener información sobre un evento ocurrido en uno de los equipos que están prestando servicio al Usuario
Descripción:	El Sistema muestra una pequeña ventana emergente en un área periférica de la pantalla con la notificación y el Usuario la señala con el ratón para leerla , o alternativamente la ignora y deja que desaparezca por sí sola
Pre-condiciones:	Sesión de monitorización iniciada por el Usuario

Curso del escenario básico	
Acción de los actores	Respuesta del sistema
	1. El Sistema muestra una ventana emergente notificando al usuario de la ocurrencia de un evento en un equipo servidor
2. El Usuario señala con el ratón el mensaje de notificación para leerlo	
3. El Usuario obtiene la información contenida en la notificación	
4. El Usuario retoma su actividad previa	
	5. El Sistema hace desparecer el mensaje de notificación de la pantalla

Curso de escenario alternativo: el Usuario ignora la notificación	
Acción de los actores	Respuesta del sistema
	1. El Sistema muestra una ventana emergente notificando al usuario de la ocurrencia de un evento en un equipo servidor
2. El Usuario ignora la notificación y continúa con su actividad previa	
	3. El Sistema hace desparecer el mensaje de notificación de la pantalla

Post-condiciones:	Ninguna

Tabla 22: CU-3 Ver notificación

Identificador:	CU-4
Nombre:	Ver historial de sesión
Actores:	Usuario
Propósito:	Revisar notificaciones recibidas que hayan podido pasar desapercibidas o ser filtradas, o para tener una visión global después del hecho
Descripción:	El Usuario abre y consulta el fichero donde se guarda el historial de notificaciones recibidas durante la sesión, y al acabar, lo cierra de nuevo. Alternativamente, puede consultar historiales de sesiones anteriores
Pre-condiciones:	Sesión de monitorización iniciada por el Usuario al menos una vez (aunque posteriormente haya sido detenida)

Curso del escenario básico	
Acción de los actores	Respuesta del sistema
1. El Usuario abre el fichero de historial de la sesión	
2. El Usuario consulta la información que necesite	
3. El Usuario cierra el fichero de historial de la sesión	
Curso de escenario alternativo: el Usuario consulta el historial de una sesión anterior	
Acción de los actores	Respuesta del sistema
1. El Usuario abre el fichero de historial de la sesión antigua que desee ver	
2. El Usuario consulta la información que necesite	
3. El Usuario cierra el fichero de historial de la sesión antigua	
Post-condiciones: Ninguna	

Tabla 23: CU-4 Ver historial de sesión

Identificador:	CU-5
Nombre:	Parar
Actores:	Usuario
Propósito:	Detener la sesión de monitorización para que el equipo en el que se encuentra deje de mostrar mensajes de notificación de eventos
Descripción:	El Usuario termina el proceso de visualización en su equipo
Pre-condiciones:	Sesión de monitorización iniciada por el Usuario

Curso del escenario básico	
Acción de los actores	Respuesta del sistema
1. El Usuario termina el proceso de visualización de eventos	
	2. El Sistema da de baja la suscripción del cliente, continuando el proceso de monitorización para otros clientes
Post-condiciones:	La sesión de monitorización deja de estar iniciada

Tabla 24: CU-5 Parar

Matriz de trazabilidad

Una vez definidos los casos de uso del sistema a desarrollar, es posible establecer una relación entre los requisitos definidos durante la especificación de requisitos y los casos de uso en los que dichos requisitos son directamente aplicables. Esta correspondencia se detalla en una *matriz de trazabilidad*, que proporciona una herramienta para facilitar el seguimiento durante etapas de desarrollo posteriores de los requisitos y su estado de implementación en función de los escenarios de uso y actividades del sistema en desarrollo.

	CU-1	CU-1.1	CU-2	CU-2.1	CU-3	CU-4	CU-5
RF-1		●					
RF-2	●						
RF-3			●				●
RF-4			●				●
RF-5			●				●
RF-6					●		
RF-7				●			
RF-8				●			
RF-9						●	
RNF-1	●		●				
RNF-2	●		●				
RNF-3		●					
RNF-4			●		●		
RNF-5			●		●		
RNF-6					●		

Tabla 25: Matriz de trazabilidad

3.4 Diseño

Durante la etapa de diseño se define y documenta la arquitectura del sistema a desarrollar, incluyendo los componentes que la forman, sus interfaces y el comportamiento de éstos, proporcionando una descripción detallada del sistema para su posterior implementación.

3.4.1 Evaluación de alternativas de diseño

Como primer paso de la etapa de diseño del proyecto se evalúan diferentes alternativas posibles en cuanto a modelos de arquitectura y componentes externos existentes que puedan ajustarse a las necesidades del proyecto tal y como han sido definidas durante la fase de análisis, y servir como punto de partida para el diseño del mismo.

Arquitectura

El sistema a desarrollar debe proporcionar al usuario un medio de visualización de notificaciones (requisito RF-6) correspondientes a eventos de sistema obtenidos mediante la monitorización de sus ficheros de registro (RF-1). Según el requisito RF-3, el usuario que realiza el proceso de monitorización debe poder hacerlo desde un equipo diferente a aquel monitorizado. Según los requisitos RF-4 y RF-5, un equipo dado debe poder ser monitorizado por múltiples usuarios diferentes, y un usuario debe poder monitorizar múltiples equipos diferentes de forma simultánea.

Esto significa que el sistema a desarrollar funciona como un **sistema distribuido** de múltiples nodos. Se han considerado 3 modelos de arquitectura de sistema distribuido para este proyecto:

- Cliente-Servidor
- Par a Par
- Publicación-Suscripción

El modelo **Cliente-Servidor** divide en dos roles diferenciados los nodos del sistema; unos proporcionan un servicio determinado (servidores) y otros solicitan y reciben dicho servicio (clientes).

En el modelo Cliente-Servidor básico, el Servidor espera solicitudes de conexión de forma continua, esperando Clientes que requieran el servicio ofrecido, normalmente en la forma de datos almacenados o generados en el Servidor, o acciones a realizar a petición de un Cliente. Cuando un Cliente requiere dicho servicio, establece una conexión con el Servidor, éste

proporciona el servicio en cuestión, y una vez finalizado, se termina la conexión.

Por regla general, un sistema Cliente-Servidor se implementa de manera que el Servidor pueda atender simultáneamente las solicitudes de más de un Cliente de forma concurrente.

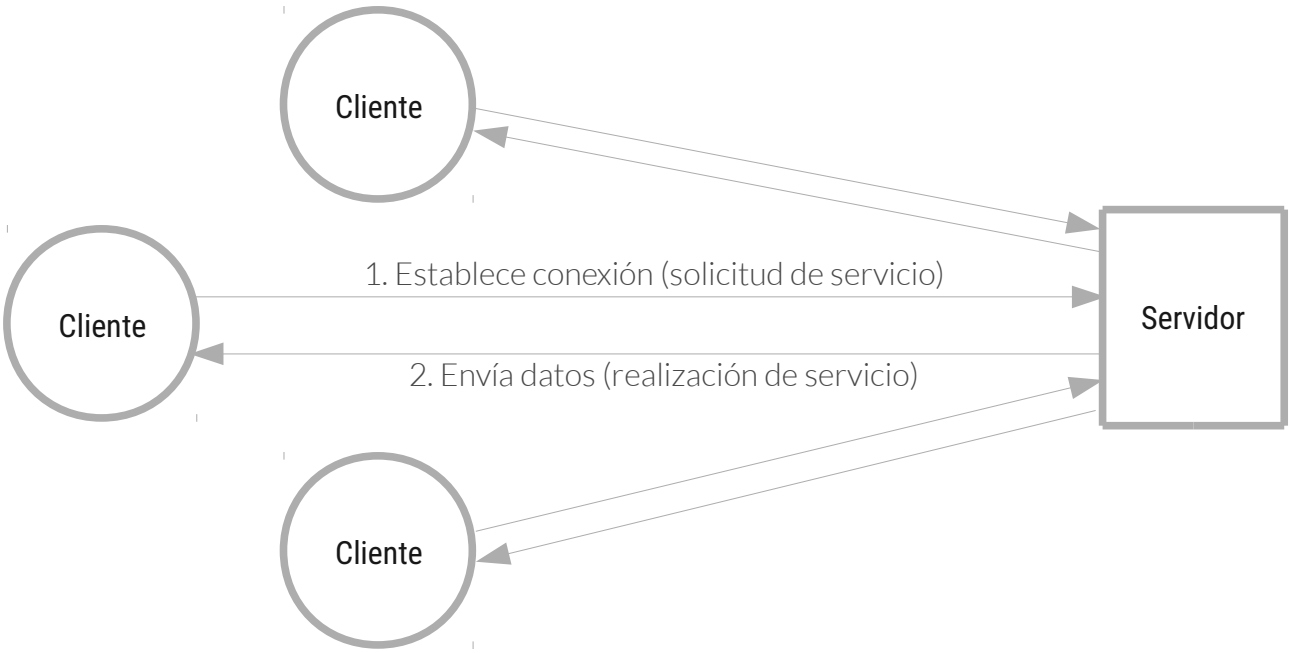

Ilustración 11: Modelo Cliente-Servidor

El modelo **Par a Par** elimina la distinción entre el rol de servidor y el de cliente, y cada nodo se comunica con los otros nodos tanto proporcionando servicio, como recibiéndolo, dando lugar a una arquitectura descentralizada. También es posible utilizar servidores cuya finalidad sea facilitar las comunicaciones entre los nodos (Buscador de Servicios), dando lugar a una arquitectura semi-centralizada.

En este modelo, cada nodo (con o sin la ayuda de un buscador de servicios) está conectado a varios otros nodos que forman parte del sistema. La funcionalidad para establecer la conexión, dar servicio a otros nodos, o recibirlo de otros nodos, están embebidos en la propia aplicación.

En el modelo de Par a Par descentralizado, además, los propios nodos tienen la funcionalidad para encaminar los datos y establecer las conexiones de uno a otro:

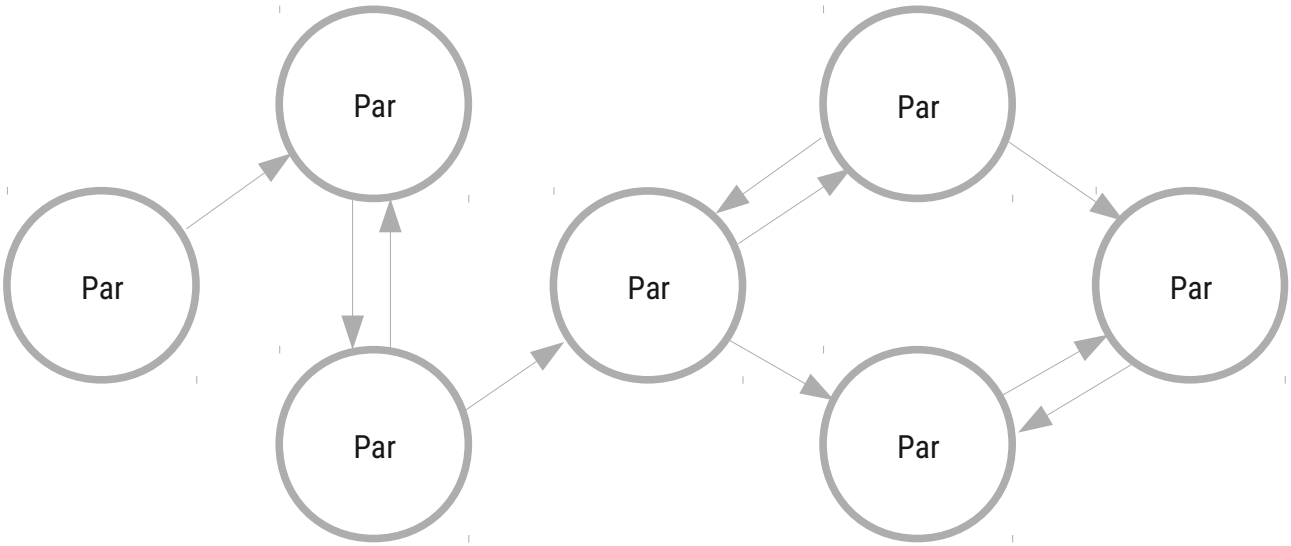

Ilustración 12: Modelo Par a Par descentralizado

En el modelo de Par a Par semi-centralizado, el servidor Buscador de Servicios actúa como asistente para facilitar el encaminamiento de datos y las conexiones entre nodos:

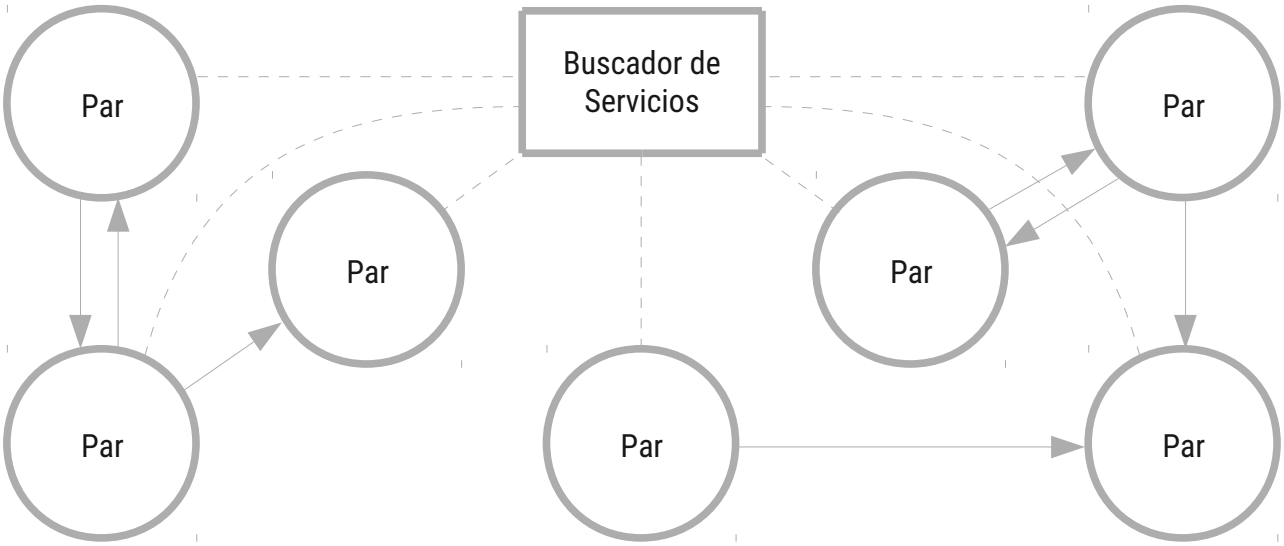

Ilustración 13: Modelo Par a Par semi-centralizado

Por último, el modelo de **Publicación-Suscripción** distingue (al igual que el modelo Cliente-Servidor) dos roles diferenciados entre los nodos, en este caso el Editor, que proporciona contenidos, y el Suscriptor, que los consume.

A diferencia del modelo Cliente-Servidor, éste se centra en el contenido en sí, y abstrae los Suscriptores para los Editores (los contenidos se publican indistintamente a, y sin tener conocimiento de, los Suscriptores que estén conectados) y los Editores para los Suscriptores (que se suscriben para recibir el contenido de interés sin tener conocimiento de los Editores que puedan o no estar conectados).

Para ello se utiliza un servidor intermediario que implementa un sistema de mensajes basado en canales de comunicación: los Suscriptores pueden suscribirse a los canales en que estén interesados, y los Editores publican contenidos en los canales apropiados. Estos canales pueden utilizarse para filtrar los contenidos por tipo, y ocasionalmente se añaden filtros adicionales sobre los contenidos para permitir definir a los clientes con una granularidad más fina qué contenidos quieren recibir.

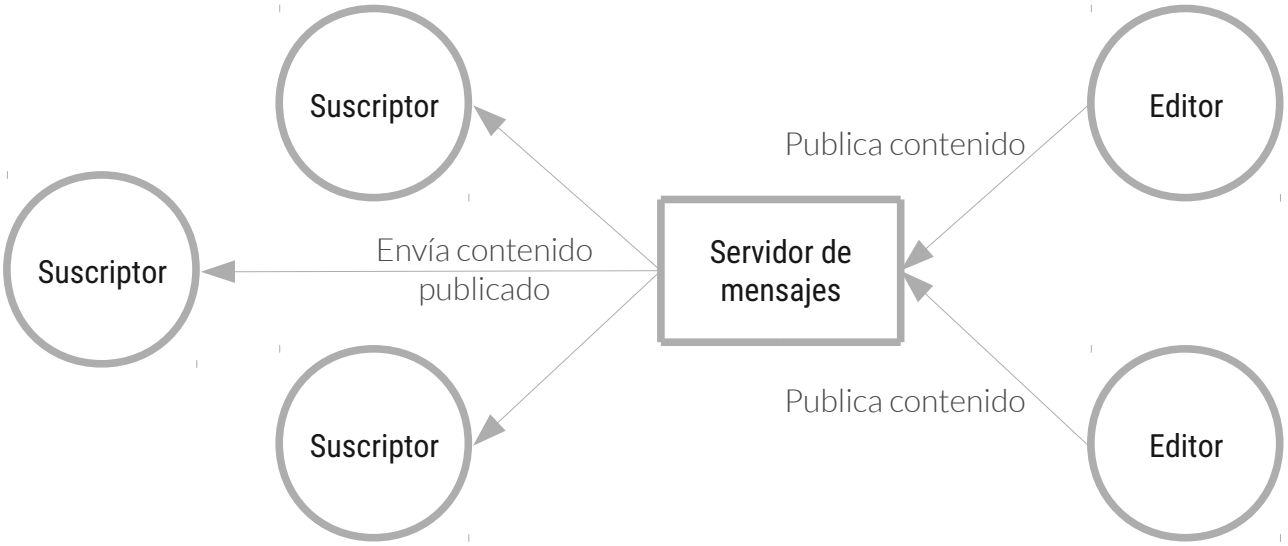

Ilustración 14: Modelo Publicación-Suscripción

A continuación se muestra una tabla comparativa con las características de cada modelo según las necesidades del sistema a desarrollar:

	Cliente-Servidor	Par a Par	Publicación-Suscripción
Conexión directa entre nodos			
Separación de roles entre los nodos			
Permite la conexión a un nodo específico			
Recibir contenido sin petición explícita			
Eficiente en cuanto al tráfico de red			

Característica disponible
Característica disponible de forma parcial o con limitaciones
Característica no disponible

Tabla 26: Tabla comparativa (modelos de arquitectura)

Como puede observarse, ninguna de las arquitecturas contempladas cubre por completo las necesidades del sistema a desarrollar. El modelo Par a Par era originalmente una opción por su topología (múltiples nodos conectados, cada uno a varios) pero una evaluación de los resultados del Análisis permite comprobar una clara diferenciación de roles en el sistema: equipos monitorizados que envían eventos y equipos monitor que los reciben y los presentan al usuario (RF-3, casos de uso CU-1 Puesta en marcha del servicio y CU-2 Suscripción).

El modelo Cliente-Servidor por otro lado encaja perfectamente con lo anterior, y permite la conexión de los procesos Cliente (que reciben los mensajes de monitorización) a los Servidores a los que se quiera conectar (CU-2 Suscripción) pero el problema fundamental con este modelo es que el Cliente **no sabe** cuándo se producen eventos de conexión (es el Servidor el que los genera cuando ocurren y los envía, RF-6). Este comportamiento puede conseguirse si el Cliente solicita actualizaciones de estado del Servidor con suficiente frecuencia, pero eso da lugar a un modelo de comunicación terriblemente ineficiente para este caso.

El modelo Publicación-Suscripción resuelve este problema, ya que con su modelo de comunicación de conexión persistente (suscripción) del Suscriptor al servidor de mensajes, éste puede enviar contenidos a los Suscriptores cuando los Editores los publican, sin una petición explícita por parte de los Suscriptores. Este modelo de comunicación encaja con el

requerido por el problema, y es por tanto una solución mucho más eficiente, pero la conexión indirecta entre unos y otros no facilita la suscripción a Editores (equipos monitorizados) específicos.

Una solución intermedia entre las dos anteriores, un Cliente-Servidor en el que los Clientes se suscriban a uno o varios Servidores directamente, estableciendo una conexión persistente y permitiendo a los Servidores enviar mensajes de monitorización cuando se produzcan eventos siguiendo un modelo de comunicación *push* análogo al de Publicación-Suscripción parece la opción más adecuada.

Componentes externos

Tal y como se detalla en el apartado 2.3, existe un gran abanico de posibilidades en cuanto a tecnologías disponibles que pueden ser utilizadas a modo de componentes del sistema para realizar algunas tareas del mismo (en particular, la monitorización de ficheros, la notificación de eventos al usuario, y la comunicación de red entre los distintos nodos del sistema).

A continuación se procede a evaluar dichas alternativas en función de sus características y las necesidades del proyecto tal y cómo se han definido durante la fase de análisis.

Monitorización de ficheros

Las alternativas contempladas para realizar el proceso de monitorización de ficheros son el uso directo de *E/S de ficheros POSIX*, los programas *tail* y *tailf*, y el subsistema del *kernel inotify*. Información detallada sobre cada uno de éstos se incluye en el apartado 2.3.1.

Según los requisitos RF-1, RF-2 y RF-6 el monitor de ficheros debe seguir múltiples ficheros obteniendo el texto añadido cada vez que se registre una nueva entrada en él y siendo capaz de continuar su funcionamiento aún cuando el sistema realice operaciones de mantenimiento con los ficheros (renombrar, eliminar, truncar…).

También son de importancia características como la facilidad para integrarlo en el resto del sistema, el no imponer dependencias externas adicionales al proyecto, la existencia de experiencia de desarrollo previa con la tecnología, el soporte en diferentes plataformas y distribuciones de GNU/Linux, el tipo de licencia o la disponibilidad de código fuente.

A continuación se muestra una tabla comparativa en base a estos criterios:

	E/S de ficheros POSIX	tail / tailf	inotify
Avisa de cambios en ficheros sin *polling*			
Proporciona texto añadido a un fichero			
Soporta operaciones de rotación			
Fácil integración con otros componentes			
No requiere otras dependencias			
Hay experiencia previa de desarrollo			
Soporte en distintas plataformas Linux			
Código fuente disponible			
Coste de la licencia (gratuita a costosa)			

Característica disponible
Característica disponible de forma parcial o con limitaciones
Característica no disponible

Tabla 27: Tabla comparativa (monitorización de ficheros)

Como puede observarse, la única alternativa que ofrece toda la funcionalidad requerida pre-implementada es la de los programas *tail* y *tail -f*. Sin embargo, al ser programas externos se introduce la dependencia de los mismos, y sobre todo, se dificulta mucho la integración con otras partes del sistema, y proporciona una solución que podría estimarse como "poco elegante".

El uso de *E/S de ficheros POSIX* no proporciona ningún medio eficiente de conocer eventos relacionados con los ficheros cuando estos se produzcan sin tener que hacer peticiones de actualización de estado continuas (*polling*), por lo que no es una solución eficiente.

La alternativa restante, *inotify*, proporciona la capacidad de recibir notificación de cambios en los ficheros seguidos procedente del sistema de ficheros cada vez que estos sean

modificados o se produzcan ciertos eventos, pero es una funcionalidad de más bajo nivel que *tail* y *tailf* y no proporciona de forma directa el resto de la funcionalidad de monitorización necesaria. Sin embargo, el hecho de que permita una sencilla integración en el sistema, el no requerir de dependencias adicionales y el tener una interfaz sencilla de usar (lo que minimiza la falta de experiencia previa con la tecnología en cuestión), lo convierten en una opción mucho más apropiada.

Si bien *inotify* no proporciona una solución completa a los requerimientos para el módulo de monitorización, si proporciona una buena base sobre la que implementar el mismo.

Notificación de eventos al usuario

Las alternativas contempladas para la notificación de eventos al usuario son *Growl*, *D-Bus / Desktop Notification Specification*, *libnotify* y *chrome.notifications*. Información detallada sobre cada uno de estos se incluye en el apartado 2.3.2.

Según los requisitos RF-6, RF-7, RF-8 y RNF-6 el sistema debe mostrar notificaciones textuales con el contenido de cada entrada añadida a un fichero monitorizado, de forma no intrusiva (mensajes posicionados en regiones periféricas de la pantalla y que permanezcan sólo un tiempo limitado si son ignorados), con posibilidades de configuración para el usuario en cuanto a sus preferencias de visualización (al menos tiempo que permanecen en pantalla) y con posibilidad por parte del usuario de establecer filtros para omitir selectivamente parte de las notificaciones en función de su procedencia, fichero afectado o contenido.

También son de importancia características como la facilidad para integrarlo en el resto del sistema, el no imponer dependencias externas adicionales al proyecto, la existencia de experiencia de desarrollo previa con la tecnología, el soporte en diferentes plataformas y distribuciones de GNU/Linux, el tipo de licencia o la disponibilidad de código fuente.

A continuación se muestra una tabla comparativa en base a estos criterios:

	Growl	D-Bus / DNS	libnotify	Chrome
Notificaciones con contenido textual				
Notificaciones en zonas periféricas				
Notificaciones que se ocultan por sí solas				
Posibilidades de configuración				
Filtrado de notificaciones				
Fácil integración con otros componentes				
No requiere otras dependencias				
Hay experiencia previa de desarrollo				
Soporte en distintas plataformas Linux				
Código fuente disponible				
Coste de la licencia (gratuita a costosa)				

Característica disponible
Característica disponible de forma parcial o con limitaciones
Característica no disponible

Tabla 28: Tabla comparativa (notificación de eventos)

Todas las opciones consideradas ofrecen buenas características de notificación y son en gran medida equiparables en ese sentido. Ninguna de ellas ofrece una solución para el filtrado de notificaciones, por lo que esa parte será necesario implementarla como otro componente separado. La más sencilla de integrar y utilizar es libnotify, y todas ellas suponen introducir dependencias adicionales en el proyecto, aunque en este caso la que sale peor parada es *chrome.notifications*, que depende del navegador de Google *Chrome* para instalar y ejecutar la aplicación final al requerir su uso desde una *Chrome App*. Las opciones multi-plataforma extra

que proporciona no son una ventaja grande ya que el objetivo es únicamente GNU/Linux; podría haber sido una consideración positiva, pero no a costa de asumir otras penalizaciones. Donde *Growl* sale peor parado es en el soporte de Linux (es originalmente de Mac OS, y las versiones para otros sistemas son de otros desarrolladores, mal documentadas y en definitiva, poco fiables – si bien aparentemente funcionales), y la licencia de pago, no tanto a nivel de desarrollo sino de usuario final (la aplicación necesaria para desplegar las notificaciones *Growl* es de pago, por lo que cualquiera que quisiera usar el sistema tendría que comprarla).

En definitiva, la mejor opción en GNU/Linux es *Desktop Notifications Specification*, ya sea directamente a través de *D-Bus*, o a través de su implementación de más alto nivel *libnotify*. **Dado que no supone ninguna desventaja para este proyecto, y resulta más sencilla de utilizar, se considera la opción de *libnotify* como la mejor para componente de notificación**.

Transmisión de eventos

Las alternativas contempladas para la transmisión por red de eventos de monitorización son *sockets POSIX*, las librerías de E/S por red *Boost.Asio*, *ENet* y *RakNet*, y el sistema de notificaciones *push* en nube *Google Cloud Messaging*. Información detallada sobre cada uno de estos se incluye en el apartado 2.3.3.

Anteriormente en la Evaluación de alternativas de diseño (en el apartado Arquitectura) ya se han evaluado ciertos aspectos relacionados con la arquitectura como sistema distribuido del proyecto. Adicionalmente, según el requisito RNF-4 el sistema utilizado debe proporcionar garantía de entrega y corrección de datos (a través de TCP u otro protocolo que cumpla el requerimiento).

También son de importancia características como la facilidad para integrarlo en el resto del sistema, el no imponer dependencias externas adicionales al proyecto, la existencia de experiencia de desarrollo previa con la tecnología, el soporte en diferentes plataformas y distribuciones de GNU/Linux, el tipo de licencia o la disponibilidad de código fuente.

A continuación se muestra una tabla comparativa en base a estos criterios:

	Sockets POSIX	Boost.Asio	ENet	RakNet	Google Coud Messaging
Conexión por LAN o Internet					
Características orientadas a conexión					
Flexibilidad en cuanto a la arquitectura					
Fácil integración con otros componentes					
No requiere otras dependencias					
Hay experiencia previa de desarrollo					
Soporte en distintas plataformas Linux					
Código fuente disponible					
Coste de la licencia (gratuita a costosa)					

Característica disponible
Característica disponible de forma parcial o con limitaciones
Característica no disponible

Tabla 29: Tabla comparativa (transmisión de eventos)

Google Cloud Messaging es una tecnología muy potente, de alto nivel, que ofrece una gran funcionalidad pero su modelo arquitectural no encaja completamente con las necesidades del proyecto (es, de hecho, una implementación del modelo de Publicación-Suscripción, discutido con anterioridad), requiere que los mensajes se reciban en una aplicación *Android* o *iOS* o en su defecto, una *Chrome App* (que sería la única opción viable, pero impone fuertes dependencias y restricciones adicionales) y presenta ciertas dificultades adicionales durante el desarrollo y posterior utilización del prototipo (teniendo que obtener una cuenta de desarrollador de Google, y registrar la aplicación con el servicio ofrecido, por ejemplo).

De entre las otras alternativas, todas ofrecen la funcionalidad necesaria. Exceptuando los *sockets POSIX* (accesibles mediante llamadas de sistema), todas introducen la dependencia de

una librería externa, aunque tampoco es un problema importante; a cambio, las librerías ofrecen APIs de más alto nivel y con algunas funciones pre-implementadas (especialmente *RakNet*, pues *Boost.Asio* es más una delgada capa sobre el sistema de conexión de sockets adaptado a C++, y *ENet* un punto intermedio, pero también de bastante bajo nivel).

Sin embargo muchas de estas funciones no son realmente requeridas por el sistema a desarrollar y **la experiencia previa en desarrollo con *sockets POSIX* se considera un criterio de peso a la hora de decidirse por estos como la mejor opción**.

Otras consideraciones

En base a las evaluaciones anteriores, se deciden los siguientes aspectos adicionales:

- **El sistema se implementará en lenguaje C++11 (ISO/IEC 14882:2011)** [45]. Tanto C como C++ son igualmente apropiados para trabajar con funciones del SO, C++ proporciona herramientas adicionales como orientación a objetos, la Librería Estándar de C++ con contenedores genéricos y otras facilidades, y C++11 proporciona varios extras de interés como las librerías de *threading* y punteros inteligentes. A día de hoy, la mayoría de compiladores tienen soporte completo de dicha revisión del estándar, como el compilador que se utilizará en el proyecto, **GNU g++ 4.9.2**.
- Se hará uso de la herramienta **Make** para la compilación y enlazado automático del código del proyecto mediante *Makefiles*.
- Se hará uso de **Doxygen** para la generación automática de documentación a partir de la implementación.
- La plataforma de desarrollo y pruebas será **GNU/Linux Debian 8.1 Jessie**, y tendrá como objetivo de compatibilidad plataformas **GNU/Linux con kernel 2.6.13+**.
- La jerarquía de directorios del proyecto será:
  - Un directorio para la aplicación servidora (*lognotifyserv*) y un directorio para la aplicación cliente (*lognotifycli*). Cada uno contendrá los siguientes subdirectorios:
    - *src* – Ficheros de código fuente (.cpp y .h)
    - *obj* – Ficheros de código objeto generados durante el proceso de compilación
    - *bin* – Ficheros binarios ejecutables generados
    - *doc* – Documentación generada por *doxygen*.

3.4.2 Arquitectura y componentes

Una vez evaluadas las diferentes alternativas de diseño y componentes externos a utilizar, se define la arquitectura definitiva del sistema a desarrollar.

El sistema se modela como un sistema distribuido en el que los nodos pueden ser de dos tipos diferentes:

- **Servidor de notificaciones:** monitoriza los ficheros de registro del sistema en que se encuentra y envía notificaciones de cambios en dichos ficheros a todos los Clientes conectados.
- **Cliente de notificaciones:** recibe notificaciones de cambios en ficheros de registro enviadas por los Servidores a los que se encuentre conectado y los muestra al usuario.

Cada Servidor puede proporcionar servicio a múltiples Clientes, y cada Cliente puede recibir servicio de múltiples Servidores simultáneamente.

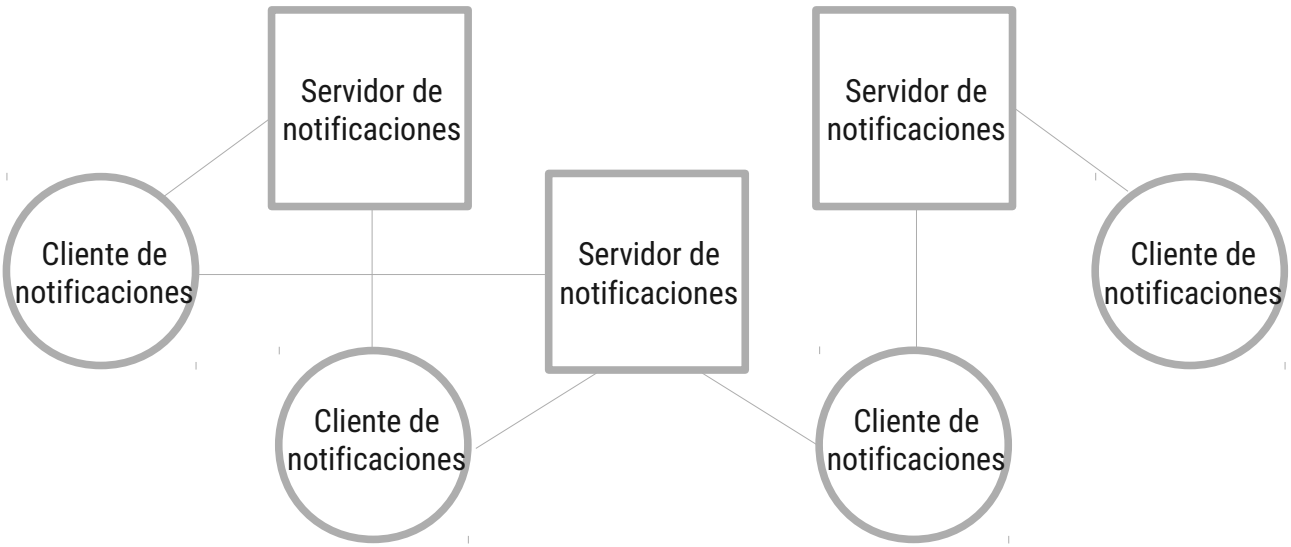

Ilustración 15: Modelo de conexión

El proceso de comunicación entre un Cliente y un Servidor comienza cuando un Cliente desea comenzar a recibir notificaciones. El Cliente se suscribe al Servidor, creando una conexión persistente con éste. A partir de ese momento, el Servidor enviará mensajes de notificación de cambios en los ficheros de registro tan pronto como éstos se produzcan a través de esta conexión. El proceso de comunicación termina con la desconexión del Cliente.

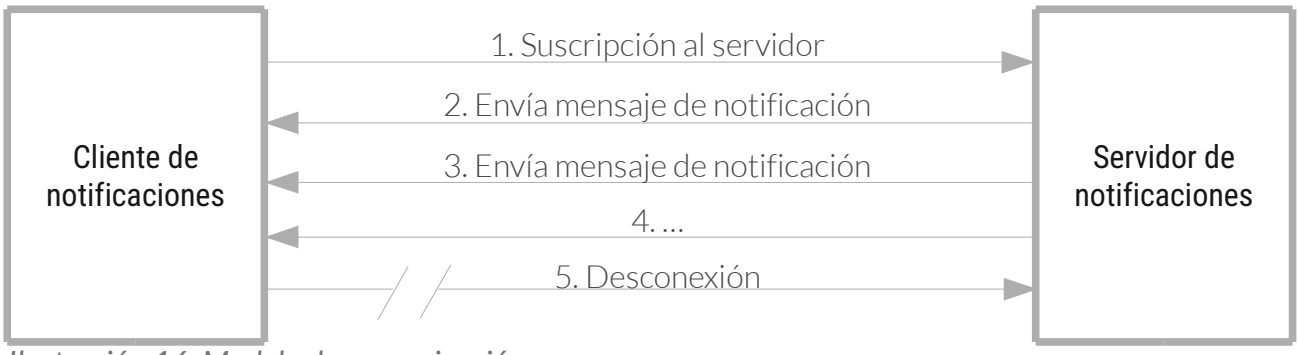

Ilustración 16: Modelo de comunicación

A continuación se muestra un diagrama de la arquitectura general del sistema, mostrando sus principales componentes, las interacciones entre los mismos y el funcionamiento general:

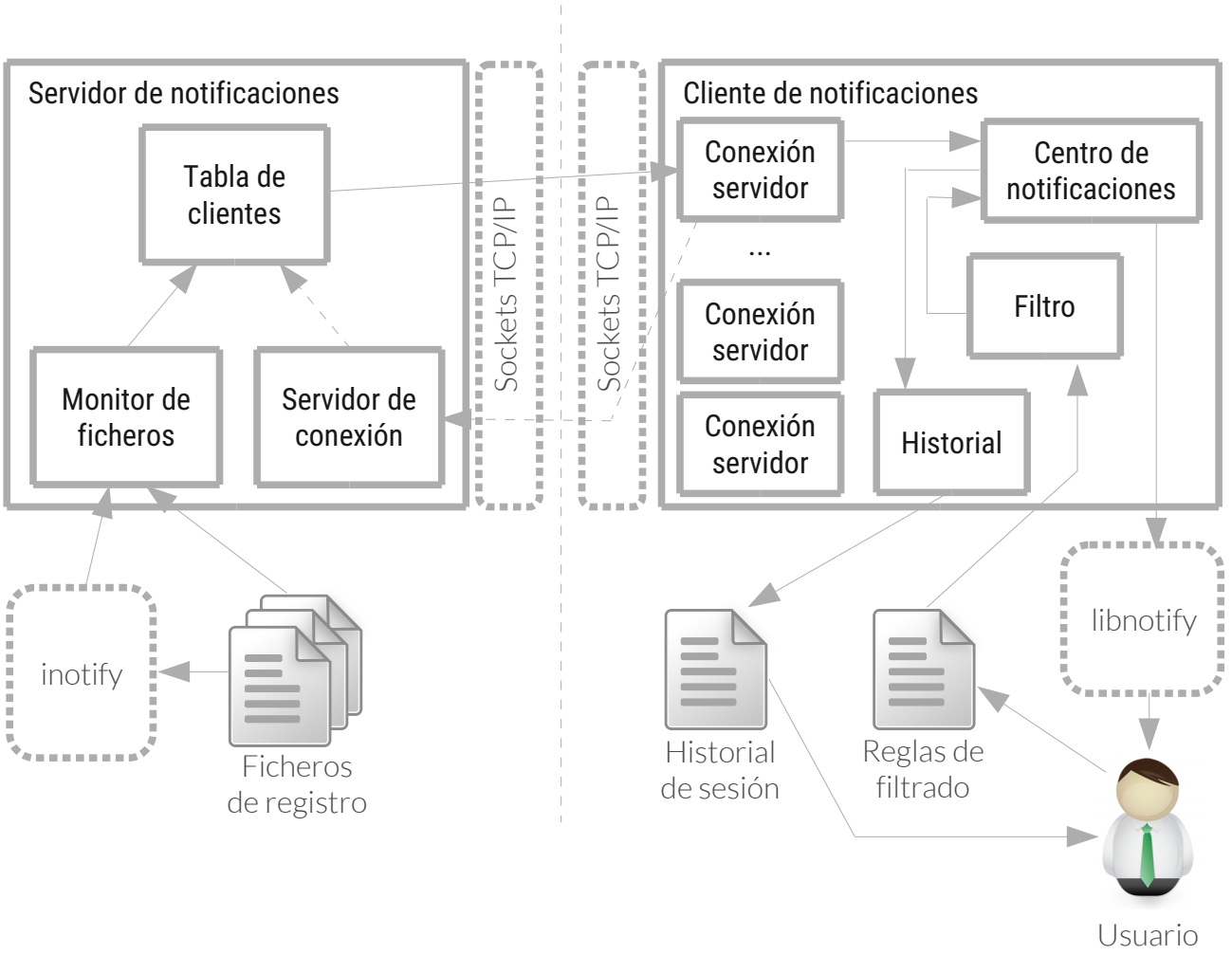

Ilustración 17: Arquitectura del sistema

Descripción breve de los componentes:

- **Servidor de notificaciones:** acepta suscripciones de Clientes y monitoriza los ficheros de registro, notificando a todos los Clientes conectados cuando se produce un evento.
- **Servidor de conexión:** acepta conexiones entrantes de Clientes para suscribirlos al servicio. Cuando se establece una conexión con un Cliente, pide a la Tabla de clientes que la añada al conjunto.
- **Tabla de clientes:** guarda todas las conexiones establecidas con Clientes, y envía mensajes de evento a cada uno de ellos cuando el Monitor de ficheros se lo indica.
- **Monitor de ficheros:** monitoriza los ficheros de registro a través de *inotify*, cada vez que se produce un evento, obtiene la información necesaria del fichero afectado y solicita a la Tabla de clientes que envíe el evento generado.
- **Cliente de notificaciones:** conecta con uno o más Servidores, y cada vez que recibe un mensaje de notificación, lo muestra al usuario mediante una notificación de escritorio.
- **Conexión servidor:** representa una conexión persistente con un Servidor, que se crea en el momento de la suscripción, y por la que recibe mensajes de notificación. Cuando esto ocurre, la pasa al Centro de notificaciones para que la procese. Puede haber múltiples Conexiones de servidor activas en el mismo Cliente.
- **Centro de notificaciones:** procesa las notificaciones recibidas. Primero pide al Historial que las registre, luego evalúa mediante el Filtro si debe ser mostrada u omitida, y si es lo primero, la muestra al usuario a través de la librería *libnotify*.
- **Historial:** registra en un fichero el *historial de la sesión*, es decir, todos los mensajes de notificación recibidos por el Cliente en la sesión actual para su consulta posterior.
- **Filtro:** evalúa si una notificación debe ser mostrada o no al usuario en función de las reglas basadas en *pattern matching* con expresiones regulares definidas por éste en un fichero de Reglas de filtrado.

Matriz de trazabilidad

Una vez definida la arquitectura general del sistema y los componentes principales que lo forman, se establece una relación entre los requisitos definidos durante el análisis (apartado 3.3.1) y los componentes del sistema que cubren cada uno de ellos. Esta correspondencia se detalla en una *matriz de trazabilidad*, que permite asegurar que todos los requisitos han sido contemplados, y qué compone está relacionado con cada requisito.

	Servidor de notificaciones	Servidor de conexión	Tabla de clientes	Monitor de ficheros	Cliente de notificaciones	Conexión servidor	Centro de notificaciones	Historial	Filtro
RF-1	●			●					
RF-2	●			●					
RF-3	●	●	●		●	●			
RF-4	●	●	●						
RF-5					●	●			
RF-6					●		●		
RF-7					●				●
RF-8					●		●		
RF-9					●			●	
RNF-1	●				●				
RNF-2	●				●				
RNF-3	●			●					
RNF-4	●	●	●		●	●			
RNF-5	●	●	●		●	●			
RNF-6					●		●		

Tabla 30: Matriz de trazabilidad (componentes)

3.4.3 Modelado de clases

Una vez definida la arquitectura del sistema y sus componentes principales, obteniendo así una visión de alto nivel del funcionamiento del sistema, se refina el diseño aportando información más detallada sobre el funcionamiento del sistema. Para ello se modelan las clases que posteriormente serán implementadas en C++ para la aplicación servidora y la aplicación cliente.

A continuación se incluye la documentación generada durante este proceso, incluyendo los diagramas de clases, y una descripción de las mismas y sus métodos.

Diagrama de clases del servidor

El siguiente diagrama muestra las relaciones de clases y objetos del servidor:

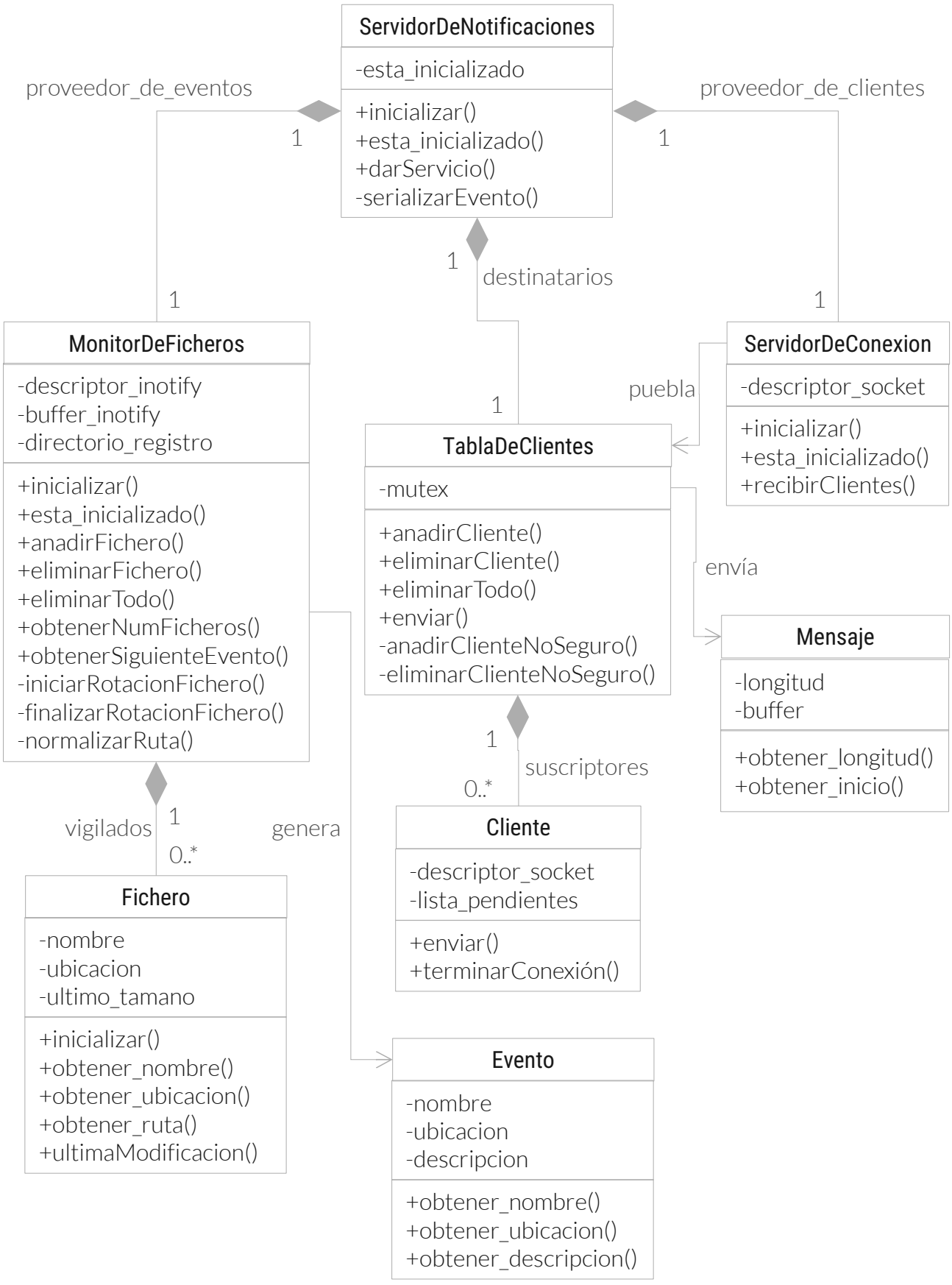

Ilustración 18: Diagrama de clases (servidor)

Descripción de clases del servidor

A continuación se incluye una descripción de la función de las clases anteriores y de su interfaz (operaciones públicas):

ServidorDeNotificaciones	
Clase principal de la aplicación servidora. Para utilizar el sistema, debe crearse una instancia de ServidorDeNotificaciones, que deberá ser inicializada para especificar su configuración. Con una llamada posterior a darServicio() el servidor empezará a enviar eventos de modificación en los ficheros especificados a aquellos clientes que se conecten al sistema.	
inicializar	Inicializa el servidor de notificaciones
esta_inicializado	Comprueba si el servidor ha sido ya inicializado con anterioridad
darServicio	Comienza el servicio de notificaciones remotas de modificación de ficheros de registro
serializarEvento	Convierte un Evento de monitorización en un Mensaje listo para ser enviado por red

Tabla 31: Clase ServidorDeNotificaciones (descripción)

MonitorDeFicheros	
Un objeto de tipo MonitorDeFicheros vigila un conjunto de ficheros, generando eventos cada vez que a uno de los ficheros monitorizados se le añada un contenido adicional, y pudiendo capturar estos eventos mediante una llamada bloqueante: obtenerSiguienteEvento(). Un objeto de esta clase debe ser inicializado antes de que se le puedan añadir ficheros o capturar eventos.	
inicializar	Inicializa el monitor de ficheros y pone en marcha el proceso de vigilancia. Hasta que no se haya inicializado, no pueden añadirse ni eliminarse ficheros
esta_inicializado	Comprueba si la instancia de MonitorDeFicheros ya ha sido inicializada
anadirFichero	Añade el fichero especificado al conjunto de ficheros monitorizados. Es importante tener en cuenta que es necesario inicializar la instancia antes de añadir ficheros
eliminarFichero	Elimina el fichero especificado del conjunto de ficheros monitorizados
eliminarTodo	Elimina todos los ficheros del conjunto de ficheros monitorizados
obtenerNumeroDeFicheros	Obtiene el número de ficheros que están siendo activamente monitorizados en el momento de la llamada
obtenerSiguienteEvento	Obtiene el siguiente evento que se produzca en cualquiera de los ficheros contenidos. obtenerSiguienteEvento es una función bloqueante: la ejecución continuará cuando se produzca el siguiente evento que pueda devolverse desde el momento en que sea llamada
iniciarRotacionFichero	Inicia el proceso de rotación de ficheros para el fichero proporcionado, dejando el fichero inactivo a la espera de que un evento posibilite empezar a monitorizarlo
finalizarRotacionFichero	Finaliza el proceso de rotación de ficheros para el fichero especificado si este se encontraba en rotación, re-

	inicializando el proceso de monitorización del fichero nuevamente
normalizarRuta	Obtiene la ruta absoluta canónica correspondiente a la ruta introducida. Dicha ruta debe ser una ruta válida, a un directorio o fichero existente en el sistema de ficheros.

Tabla 32: Clase MonitorDeFicheros (descripción)

Fichero	
Clase que representa los ficheros de registro con los que trabaja el sistema. Cada objeto inicializado de clase Fichero contiene toda la información local relativa al propio fichero (su nombre, ubicación dentro del directorio de ficheros de registro del sistema, los últimos cambios producidos en el fichero...). Cada objeto debe de ser inicializado manualmente una vez construido antes de ser funcional, pues dicha inicialización puede fallar.	
inicializar	Carga los datos asociados al fichero especificado
obtener_nombre	Devuelve el nombre del fichero
obtener_ubicacion	Devuelve la ruta local del directorio en que se ubica el fichero, en relación al directorio de registros del sistema
obtener_ruta	Devuelve la ruta completa del fichero relativa al directorio de registros del sistema
ultimaModificacion	Devuelve el contenido añadido al fichero desde la última vez que se empleó esta función. Si es la primera vez que se utiliza, devuelve el contenido añadido desde su inicialización NOTA: es posible que el fichero haya sido rotado/eliminado/truncado/vaciado desde su último acceso, en cuyo caso esta función retornará una cadena vacía ("").

Tabla 33: Clase Fichero (descripción)

ServidorDeConexion	
Un objeto ServidorDeConexion permite aceptar conexiones entrantes y poblar una TablaDeClientes conforme vayan llegando estas. El ServidorDeConexion trabaja en un hilo aparte de forma que la aplicación no quede bloqueada esperando nuevas conexiones entrantes. Un objeto ServidorDeConexion debe de ser inicializado después de su construcción con inicializar() antes de comenzar a recibir conexiones.	
inicializar	Inicializa el ServidorDeConexion con los valores especificados, quedando preparado para empezar a recibir nuevos clientes. NOTA: Es necesario ejecutar esta función antes de empezar a recibir clientes.
estaInicializado	Indica si la instancia del ServidorDeConexion ya ha sido correctamente inicializada
recibirClientes	Crea un nuevo hilo de ejecución en el que el servidor escucha nuevas conexiones entrantes para aceptar nuevos clientes y añadirlos a la TablaDeClientes. El ServidorDeConexion debe haber sido previamente inicializado correctamente; en caso contrario, recibirClientes retornará un error

Tabla 34: Clase ServidorDeConexion (descripción)

TablaDeClientes	
Una TablaDeClientes representa una colección de clientes conectados a los que se puede enviar mensajes de red. La TablaDeClientes es *thread safe*, permitiendo ser accedida desde distintos hilos (por ejemplo para ser poblada desde un ServidorDeConexion en un hilo y enviarse mensajes a los clientes ya registrados en otro) con seguridad.	
anadirCliente	Añade un nuevo Cliente a la TablaDeClientes a partir de una conexión creada para dicho cliente.
eliminarCliente	Elimina un Cliente de la TablaDeClientes, terminando la conexión si todavía está abierta
eliminarTodo	Elimina todos los Clientes de la TablaDeClientes, terminando la conexión de cada uno si todavía está abierta
enviar	Envía un Mensaje a TODOS los clientes registrados (broadcast). Si alguna de las conexiones con estos clientes se ha perdido, estos serán eliminados automáticamente
anadirClienteNoSeguro	Añade un nuevo Cliente a la TablaDeClientes a partir de una conexión creada para dicho cliente. Esta versión de la función miembro anadirCliente NO es segura para acceso concurrente, por lo que sólo debe de ser llamada desde el interior de una sección crítica ya asegurada. Su uso está pensado únicamente para uso interno de objetos de la propia clase TablaDeClientes.
eliminarClienteNoSeguro	Elimina un Cliente de la TablaDeClientes, terminando la conexión si todavía está abierta. Esta versión de la función miembro eliminarCliente NO es segura para acceso concurrente, por lo que sólo debe de ser llamada desde el interior de una sección crítica ya asegurada. Su uso está pensado únicamente para uso interno de objetos de la propia clase TablaDeClientes

Tabla 35: Clase TablaDeClientes (descripción)

Cliente	
Cada Cliente es una abstracción de una conexión con un cliente del sistema, que permite enviar objetos de tipo Mensaje a dicho cliente a través de la conexión TCP/IP creada. Crear y aceptar dicha conexión no es responsabilidad de la clase Cliente; una vez creada la conexión, el descriptor de fichero del socket será pasado al constructor de esta clase para crear la abstracción del cliente en torno al mismo	
enviar	Envía el Evento especificado al cliente. El mensaje es enviado de forma asíncrona, de forma que una vez que se ha llamado a esta función el mensaje queda en proceso de ser enviado a su destino, pero esto puede ocurrir de forma no inmediata. La ejecución puede continuar normalmente como si el mensaje hubiera sido enviado (lo que ocurrirá tan pronto como sea posible)
terminarConexion	Termina la conexión con el cliente si esta aún está vigente y cierra el socket de conexión. Es importante terminar la conexión con el Cliente mediante una llamada a esta función, pues destruir el objeto no lo hace automáticamente. Esto es así para permitir la duplicación o movimiento de objetos Cliente sin terminar la conexión para todos ellos cuando uno de ellos es destruido.

Tabla 36: Clase Cliente (descripción)

Evento	
Un objeto de tipo Evento es generado cada vez que un fichero monitorizado es modificado. El evento contiene los datos relativos al mismo, incluyendo la ubicación completa y nombre del fichero, así como una descripción textual del evento (que normalmente contendrá el texto añadido en la modificación).	
obtener_nombre	Obtiene el nombre del fichero que ha provocado el evento
obtener_ubicacion	Obtiene la ruta completa del directorio en que se encuentra el fichero que ha provocado el evento
obtener_descripcion	Obtiene la descripción textual del evento provocado

Tabla 37: Clase Evento (descripción)

Mensaje	
La clase Mensaje encapsula un buffer con datos para enviar mediante una comunicación de red asegurando que su contenido permanezca inalterable hasta su destrucción	
obtener_longitud	Devuelve la longitud en bytes del mensaje
obtener_inicio	Devuelve un puntero al primer byte del mensaje

Tabla 38: Clase Mensaje (descripción)

Diagrama de clases del cliente

El siguiente diagrama muestra las relaciones de clases y objetos del cliente:

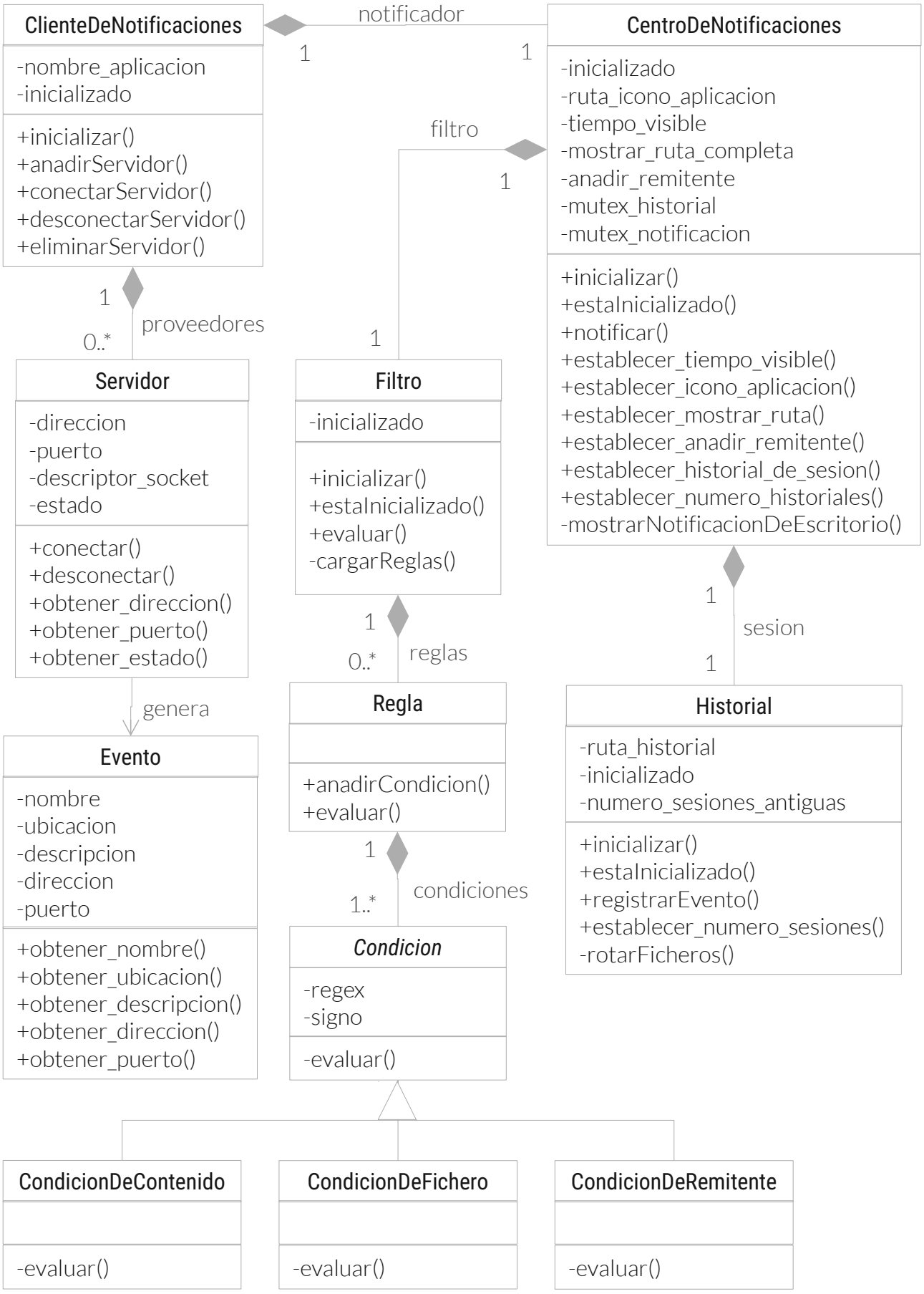

Ilustración 19: Diagrama de clases (cliente)

Descripción de clases del cliente

A continuación se incluye una descripción de la función de las clases anteriores y de su interfaz (operaciones públicas):

ClienteDeNotificaciones	
Clase principal de la aplicación cliente del sistema, desde la que se controlan las funcionalidades de la misma. Una instancia de ClienteDeNotificaciones debe de ser inicializada mediante la función inicializar() para empezar a funcionar	
inicializar	Obtiene el nombre del fichero que ha provocado el evento
anadirServidor	Registra un nuevo Servidor. Esta llamada no conecta automáticamente a dicho Servidor, para ello deberá hacerse uso de otra llamada posterior a conectarServidor() Dirección IP del servidor Puerto TCP en el que el servidor espera la conexión

Tabla 39: Clase ClienteDeNotificaciones (descripción)

Servidor	
Cada instancia de la clase Servidor es una abstracción de un servidor del sistema en la aplicación cliente. Permite conectar con dicho servidor, recibiendo datos del mismo de forma concurrente al hilo principal de la aplicación, así como gestionar cualquier otro tipo de interacción con el mismo, incluyendo la obtención de los datos del servidor o su estado.	
conectar	Solicita una conexión al servidor. En caso de tener éxito, el Servidor comenzará a transmitir eventos al CentroDeNotificaciones especificado. Este proceso se lleva a cabo de forma concurrente al hilo principal de la aplicación, por lo que ésta puede proseguir normalmente una vez realizada esta llamada
desconectar	Termina la conexión con el servidor en caso de que exista una
obtener_estado	Devuelve el estado de conexión actual del servidor
obtener_direccion	Devuelve la dirección IP del servidor
obtener_puerto	Devuelve el puerto TCP a través del que se establece la conexión con el servidor

Tabla 40: Clase Servidor (descripción)

CentroDeNotificaciones	
Un CentroDeNotificaciones procesa las solicitudes de notificación de eventos, presentando en pantalla las notificaciones de escritorio correspondientes. También permite establecer las opciones de configuración para dicha tarea. Una instancia de CentroDeNotificaciones debe de ser correctamente inicializada como paso previo a ejercer su función.	
inicializar	Inicializa la instancia de CentroDeNotificaciones con los parámetros iniciales, preparándola para presentar notificaciones de escritorio. NOTA: No puede empezar a utilizarse hasta después de haber sido inicializado y no puede inicializarse más de una vez (una segunda llamada a esta función devolverá error)
estaInicializado	Chequea si la instancia de CentroDeNotificaciones ya ha sido inicializada correctamente
notificar	Procesa un Evento para notificarlo. Esta función puede utilizarse de forma segura desde varios hilos diferentes.
establecer_tiempo_visible	Especifica el tiempo que las notificaciones de escritorio permanecen en pantalla una vez mostradas
establecer_icono_aplicacion	Especifica el icono por defecto de la aplicación que aparecerá en las notificaciones de escritorio
establecer_mostrar_ruta_completa	Especifica si se desea que en la cabecera de las notificaciones se muestre la ruta completa del fichero o solamente el nombre del mismo.
establecer_anadir_remitente	Especifica si se desea añadir el servidor que ha enviado la notificación (remitente) al final de la notificación.

establecer_historial_de_sesion	Especifica si se desea guardar un historial de eventos para la sesión (por defecto no). En caso de que se utilice esta función, se activará el uso del historial de sesión que se guardará en el fichero con la ruta especificada
establecer_numero_de_historiales_antiguos	Especifica el número de historiales de sesiones anteriores que se preservarán por seguridad en caso de que se utilice el Historial de sesión. Es importante observar que esta función debe de ser llamada ANTES de activar el uso del Historial de sesión, pues es durante el proceso de inicialización cuando realiza las rotaciones apropiadas de ficheros para salvaguardar las sesiones antiguas
mostrarNotificacionDeEscritorio	Presenta una notificación de escritorio en pantalla

Tabla 41: Clase CentroDeNotificaciones (descripción)

Historial	
Una instancia de Historial permite crear un historial de todas las notificaciones recibidas a lo largo de la sesión en un fichero que pueda ser revisado posteriormente. El historial permite mantener un cierto número de sesiones pasadas almacenadas, a partir del cual las sesiones más antiguas se irán eliminando	
inicializar	Inicializa la instancia de Historial, preparándola para registrar notificaciones. También rota los ficheros de sesiones previos de acuerdo a los parámetros introducidos
estaInicializado	Indica si la instancia de Historial ya ha sido correctamente inicializada
registrarEvento	Registra un evento de notificación en el historial
establecer_numero_sesiones_antiguas	Establece cuantas sesiones antiguas se mantendrán en la rotación de ficheros de Historial
rotarFicheros	Rota los ficheros de sesiones antiguas de forma que se preserven hasta el número indicado por parámetros de las mismas, y cualquier historial de sesión adicional sea eliminado. Se preservan los N ficheros de historial de sesiones antiguas más recientes, numerados con .1, .2, .3, etcétera hasta el número indicado por orden de más reciente a más antiguo.

Tabla 42: Clase Historial (descripción)

Filtro	
Un objeto de tipo Filtro se utiliza para filtrar eventos antes de notificarlos al usuario, de forma que no todos los eventos provoquen una notificación de escritorio. De esta forma, un usuario puede filtrar parte de los eventos producidos, de forma que no sea importunado por eventos que no quiera ver. Las reglas que componen el Filtro se definen en un fichero que el propio Filtro parsea durante su inicialización, y cada una se compone de una serie de condiciones consistentes en comparaciones de alguno de los campos del Evento con expresiones regulares. Un objeto de tipo Filtro debe de ser inicializado para estar activo	
inicializar	Inicializa y activa el Filtro, con las reglas definidas en el fichero cuya ruta se pasa por parámetro
estaInicializado	Indica si la instancia de Filtro ya ha sido correctamente inicializada
evaluar	Evalúa si un Evento debe ser notificado al usuario (es decir, si pasa el Filtro) o no. La evaluación resulta positiva si el Evento no dispara ninguna de las reglas de omisión que conforman el Filtro (es decir, si al menos una Regla se cumple para el Evento en cuestión, no pasará el Filtro)
cargarReglas	Extrae las reglas definidas en el fichero indicado, cargándolas en el Filtro

Tabla 43: Clase Filtro (descripción)

Regla	
Cada instancia de Regla define una regla del Filtro de notificaciones. Una Regla está compuesta por un conjunto de condiciones lógicas cada una de las cuales puede evaluar a true o false. Una Regla evalúa a true cuando TODAS las condiciones que la componen evalúan a true, false en caso contrario. Es decir, la evaluación de la Regla es equivalente al AND (Y) lógico de todas las condiciones que la componen	
anadirCondicion	Añade una nueva condición a la Regla
evaluar	Decide si la Regla se cumple o no para el Evento especificado. Para que una regla se cumpla, todas y cada una de las condiciones que la forman deben cumplirse también (es decir, el resultado de la función evaluar() es el AND lógico de todas las condiciones que componen la regla)

Tabla 44: Clase Regla (descripción)

Condicion	
Clase abstracta que define la estructura e interfaz de una condición de una Regla para el Filtro. Una Condicion se compone de una expresión regular en formato ECMAScript y una fución (implementada en las clases derivadas) evaluar () que decide si la condición se cumple o no para una notificación	
evaluar	Función virtual que decide si la Condicion se cumple o no para el evento especificado

Tabla 45: Clase Condicion (descripción)

CondicionDeContenido	
Condición basada en el contenido (descripción) del evento. Si la descripción del evento se iguala con la expresión regular que define la instancia de CondicionDeContenido, la condición se cumple	
evaluar	Decide si la Condicion se cumple o no para el evento especificado. La condición se cumple cuando el contenido (descripción) del evento iguala la expresión regular que define la condición

Tabla 46: Clase CondicionDeContenido (descripción)

CondicionDeFichero	
Condición basada en el nombre del fichero que origina el evento. Si el nombre del fichero se iguala con la expresión regular que define la instancia de CondicionDeFichero, la condición se cumple	
evaluar	Decide si la Condicion se cumple o no para el evento especificado. La condición se cumple cuando el nombre del fichero que origina el evento iguala la expresión regular que define la condición

Tabla 47: Clase CondicionDeFichero (descripción)

CondicionDeRemitente	
Condición basada en la dirección IP del servidor que envía el evento. Si la dirección se iguala con la expresión regular que define la instancia de CondicionDeRemitente, la condición se cumple	
evaluar	Decide si la Condicion se cumple o no para el evento especificado. La condición se cumple cuando la dirección IP del servidor que envía el evento iguala la expresión regular que define la condición

Tabla 48: Clase CondicionDeRemitente (descripción)

Evento	
Un objeto de tipo Evento encapsula los datos enviados por un servidor en relación a un evento de monitorización para ser notificado al usuario, incluyendo la ubicación completa y nombre del fichero donde se ha registrado, una descripcion textual del evento, y la dirección del servidor que lo ha enviado.	
obtener_nombre	Obtiene el nombre del fichero que ha provocado el evento
obtener_ubicacion	Obtiene la ruta completa del directorio en que se encuentra el fichero que ha provocado el evento
obtener_descripcion	Obtiene la descripción textual del evento provocado
obtener_direccion	Obtiene la dirección IP del servidor que ha enviado el evento
obtener_puerto	Obtiene el puerto TCP correspondiente a la conexión con el servidor que ha enviado el evento

Tabla 49: Clase Evento (descripción)

3.5 Implementación (servidor)

A continuación se proporciona una descripción textual de los aspectos fundamentales de la implementación en C++ de la aplicación servidora del sistema desarrollado.

3.5.1 Proceso

El programa servidor está implementado de forma que actúe como un demonio, es decir un proceso ejecutado en segundo plano sin una salida directa a través de la consola de comandos, interfaz gráfica o cualquier otro medio de interacción directo con el usuario o administrador, y sin vinculación directa a la terminal que lo ejecuta.

El procedimiento seguido para ello es:

1. Crear un nuevo proceso hijo y terminar la ejecución del original
2. Cambiar la máscara de permisos de ficheros para no depender de la del proceso padre
3. Crear una nueva sesión haciendo al nuevo proceso propietario de la misma
4. Cambiar el directorio de trabajo al directorio raíz del sistema de ficheros ("/")
5. Cerrar los descriptores de fichero de salida estándar (*STDIN*, *STDOUT*, *STDERR*)

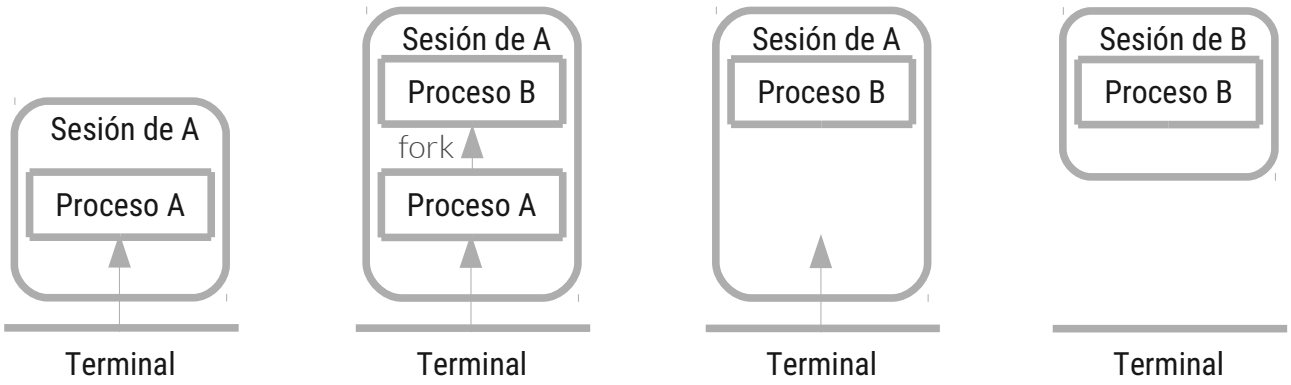

Ilustración 20: Proceso de desvinculación del demonio

La implementación de este proceso se realiza mediante el uso de las funciones de sistema *fork* (clonar proceso), *umask* (establecer máscara), *setsid* (establecer ID de sesión), y *chdir* (cambiar directorio de trabajo).

Este procedimiento se lleva a cabo en la función *main* del programa, inmediatamente después del tratamiento de parámetros de línea de comandos, y de forma condicional, dando la opción de ejecutar el programa como un proceso de terminal normal si el usuario lo requiere.

3.5.2 Seguimiento de ficheros de registro

Intervienen: *ServidorDeNotificaciones*, *MonitorDeFicheros*, *Fichero*, *Evento*

El *ServidorDeNotificaciones* es el encargado de solicitar al *MonitorDeFicheros* el seguimiento de un fichero (aquellos especificados por el usuario).

El *MonitorDeFicheros* utiliza una instancia de ***inotify*** (librería de SO) para realizar su función de seguimiento. Una instancia de *inotify* tiene asignado un descriptor de fichero que puede ser usado para leer eventos del sistema de ficheros asociados a las vigilancias (*watches*) activos. Para cada fichero o directorio que se desee seguir, se pide a *inotify* que vigile a dicho fichero o directorio. A cada *watch* de fichero o directorio se le asigna su propio descriptor de fichero, y por tanto puede ser terminada de forma independiente.

Obtener eventos de *inotify* bloquea el hilo hasta que ocurra uno (o más) eventos que notificar. En ese momento, *inotify* retorna una o más notificaciones indicando el descriptor del *watch* para el que ha ocurrido el evento y el tipo de evento. *Inotify* notifica los siguientes tipos de evento:

IN_ACCESS	El fichero es accedido
IN_ATTRIB	Cambios en los metadatos
IN_CLOSE_WRITE	Fichero abierto para escritura es cerrado
IN_CLOSE_NO_WRITE	Fichero/directorio no abierto para escritura es cerrado
IN_CREATE	Fichero/directorio creado en directorio vigilado
IN_DELETE	Fichero/directorio borrado de directorio vigilado
IN_DETELE_SELF	El fichero/directorio vigilado es borrado
IN_MODIFY	El fichero es modificado
IN_MOVE_SELF	El fichero/directorio vigilado es movido/renombrado
IN_MOVE_FROM	Fichero/directorio del directorio vigilado es movido/renombrado
IN_MOVED_TO	Fichero/directorio es movido/renombrado al directorio vigilado
IN_OPEN	El fichero/directorio es abierto

Tabla 50: Tipos de evento de inotify

Una instancia de *inotify* puede reportar eventos de uno o más tipos en particular (en caso de que no todos ellos sean relevantes para el programa que lo utiliza) por *watch* registrado. Para el seguimiento básico sólo es necesario *IN_MODIFY*, para contemplar otro tipo de situaciones (ver 3.5.3) se procesarán otro tipo de eventos.

Adición de ficheros a seguir

Cada vez que el *MonitorDeFicheros* reciba instrucción de seguir un fichero determinado:

1. Crea un objeto *Fichero* correspondiente al fichero especificado
2. Añade un *watch* a la instancia de *inotify*, obteniendo un descriptor entero del mismo
3. Inserta el objeto *Fichero* en la lista de ficheros vigilados en la posición correspondiente al número del descriptor

La lista se implementa haciendo uso de un *std::vector* disperso de punteros de propiedad no compartida (*std::unique_ptr*) a objetos *Fichero* indizados por descriptor de *watch*:

- El *vector* puede ir variando su tamaño en memoria según el número de elementos
- Los descriptores de fichero para un proceso empiezan en 1, son consecutivos y se reutilizan al ser cerrados, por lo que pese a ser disperso puede mantenerse una densidad razonable
- El número de descriptores por proceso es limitado y cada elemento es sólo un puntero por lo que el espacio máximo en memoria no puede crecer desmesuradamente
- Esto permite un acceso en tiempo constante amortizado O(1) al objeto *Fichero* cada vez que se resuelva un evento

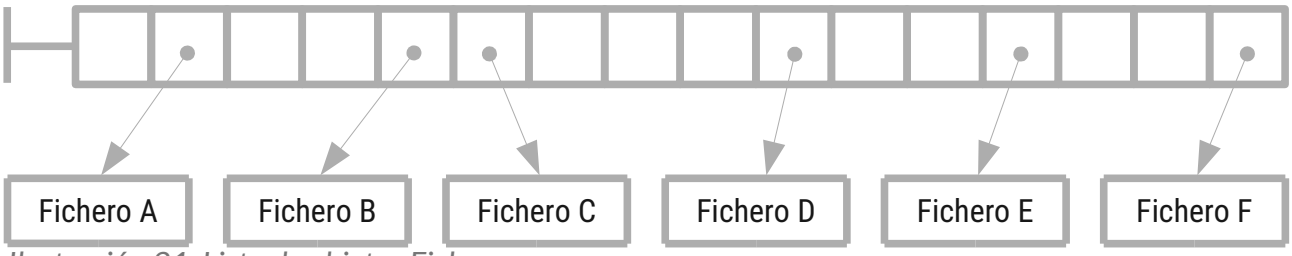

Ilustración 21: Lista de objetos Fichero

Con esta estructura, cada vez que se produzca un evento de *inotify*, se puede acceder al objeto *Fichero* correspondiente directamente por el índice y hacer con él lo que se requiera sin tener que recorrer la lista o hacer comparaciones de ningún tipo, ya que cada evento de *inotify* indica el fichero o directorio afectado a través de su descriptor.

Generación de eventos

El *MonitorDeFicheros* es el encargado de generar objetos de tipo *Evento* por cada notificación de evento de sistema de ficheros que produzca *inotify*.

El *ServidorDeNotificaciones* obtiene los *Eventos* del *MonitorDeFicheros* para enviarlos más adelante (ver 3.5.5) en un bucle continuo (espera un nuevo *Evento*, lo procesa, espera el siguiente…).

Cada lectura de *inotify* devuelve un *buffer* de caracteres que contiene uno o más eventos y el *MonitorDeFicheros* devuelve objetos de tipo *Evento* a razón de 1 por llamada, así que se utiliza un *buffer* de objetos *Evento* y en cada *obtenerEvento* el proceso es:

1. Si hay *Eventos* en el *buffer*, se devuelve el primero de ellos (terminando la función)
2. Si no hay, se lee de *inotify* bloqueando el hilo de ejecución hasta que *inotify* retorne
3. Se extraen todos los eventos de *inotify* creando los objetos *Evento* correspondientes
4. Se añaden éstos al *buffer* de *Eventos*
5. Se devuelve el primer *Evento* del buffer (terminando la función)

Para generar los objetos *Evento* se necesitan el nombre del fichero, su ubicación, y el texto añadido en la modificación (es decir, la nueva entrada añadida al fichero de registro). Todos estos datos se obtienen a partir del *Fichero* correspondiente de índice igual al descriptor de *watch* devuelto por *inotify*.

El *Fichero* almacena el nombre y la ubicación, y para obtener el texto añadido, guarda la posición de fin de fichero correspondiente a la última vez que fue consultado: extrae el texto desde esa posición hasta el final y actualiza la última posición leída.

3.5.3 Rotación y mantenimiento de ficheros de registro

Además de escribirse nuevas entradas en ellos, los ficheros de registro del sistema pueden sufrir otro tipo de cambios. Es habitual que los SSOO realicen tareas de mantenimiento como rotación y eliminación de ficheros como se explica en 2.1.3, por lo que es necesario variar el procedimiento explicado anteriormente en 3.5.2 para contemplar otro tipo de eventos.

Para tratar este aspecto, se hace uso de los eventos de *inotify* de tipo IN_MOVE_SELF e IN_DELETE_SELF.

Cuando un fichero es destruido o renombrado:

1. El *watch* de *inotify* es terminado
2. Se extrae el *Fichero* de la lista de ficheros vigilados
3. Se añade a una lista de ficheros a la espera de ser puestos nuevamente en vigilancia tan pronto como el nuevo fichero que lo sustituya (que tendrá el mismo nombre y ubicación que el anterior) sea creado
4. Se añade un *watch* para el directorio correspondiente a la ubicación del fichero si no había ya uno para esa misma ubicación, esperando por IN_CREATE o IN_MOVE_TO

La lista de ficheros a la espera se implementa con los ficheros agrupados por directorio de ubicación en dos niveles, el primero un vector disperso en el que cada índice se corresponde con el descriptor del *watch* del directorio, y el segundo, una lista densa con todos los objetos de tipo *Fichero* a la espera que comparten dicha ubicación:

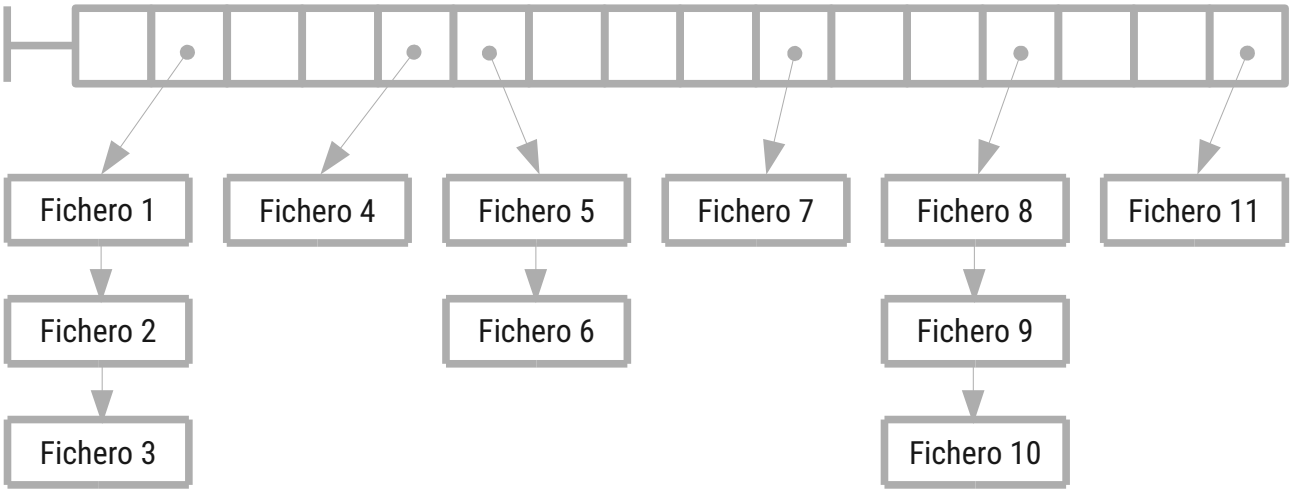

Ilustración 22: Lista de ficheros a la espera, agrupados por ubicación-directorio

Cada vez que hay que añadir un Fichero a la lista de espera, se comprueba primero si hay ya ficheros que compartan la misma ubicación en dicha lista, en cuyo caso simplemente se une a ellos, y si no, es cuando se crea un nuevo *watch* para el directorio y se añade a la lista en la posición correspondiente.

Cuando en el procesado de eventos de *inotify* se obtiene un IN_CREATE o IN_MOVE_TO:

1. Se recorre la lista de *Ficheros* a la espera en la posición correspondiente al *watch* del directorio
2. Si el nombre del nuevo fichero (IN_CREATE o IN_MOVE_TO) se corresponde con uno de los *Ficheros* a la espera de volver a ser monitorizados, se reinserta en la lista de *Ficheros* vigilados con un nuevo *watch* (ya existe, luego puede volver a ser vigilado)
3. Si el directorio en cuestión ya no tiene más ficheros a la espera, se elimina el *watch*

De esta forma el programa sólo procesa eventos de creación o movimiento de ficheros cuando hay ficheros que deben volver a ser monitorizados, y sólo para los directorios en los que esto ocurra, evitando procesar eventos innecesarios de sistema de ficheros.

El motivo por el que es necesario realizar este proceso, y no es suficiente con dejar el *watch* de fichero original, es que *inotify* realiza el seguimiento por fichero físico, a través de su inodo y no en función del nombre/ruta del mismo, por lo que el nuevo fichero de registro requiere un nuevo *watch* para ser seguido, y en el momento en el que el anterior es descartado (destruido o renombrado) no ha sido creado todavía.

3.5.4 Aceptación de solicitudes de servicio (conexiones entrantes)

Intervienen: *ServidorDeConexion*, *TablaDeClientes*

El *ServidorDeConexion* es el encargado de suministrar clientes al *ServidorDeNotificaciones* a los que enviar los *Eventos* de monitorización de ficheros generados.

El *ServidorDeConexion* está implementado con *sockets* POSIX sobre el protocolo TCP/IP para IPv4 e IPv6.

Como los clientes conectan al servidor de forma asíncrona al hilo de ejecución principal de creación y envío de *Eventos* de monitorización, el *ServidorDeConexion* acepta conexiones entrantes **en un segundo hilo de ejecución** (implementado mediante *std::thread*).

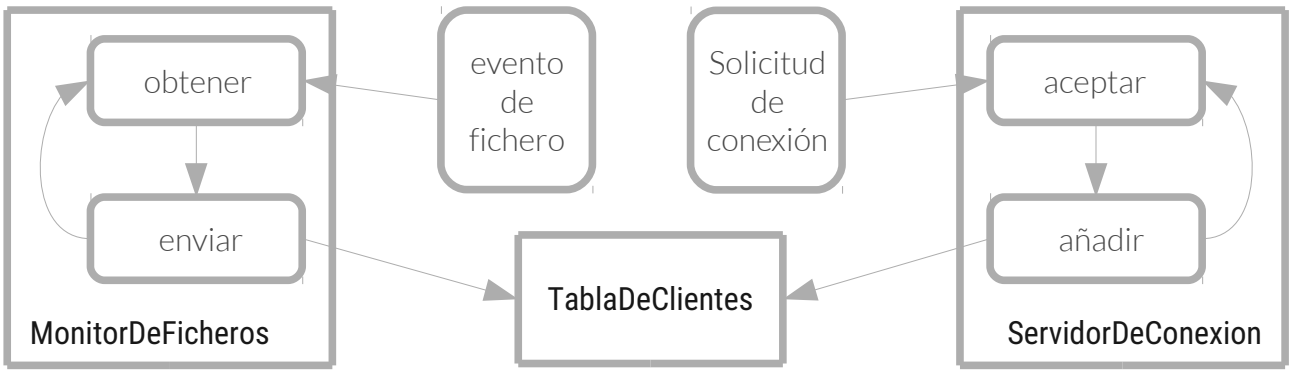

Ilustración 23: Tareas asíncronas de monitorización y aceptación de clientes

Cuando se acepta una nueva conexión, se pide a la *TablaDeClientes* que añada un nuevo *Cliente* y se le da el *socket* recién creado correspondiente a la nueva conexión para ello.

La *TablaDeClientes* gestiona toda la interacción del resto del sistema con los *Clientes*, para asegurar el correcto acceso a los mismos desde diferentes hilos. Para ello hace internamente uso de *mutex* (*std::mutex*).

3.5.5 Envío de eventos de notificación

Intervienen: *ServidorDeNotificaciones*, *TablaDeClientes*, *Evento*

Cada vez que el *ServidorDeNotificaciones* obtiene un nuevo *Evento*, éste debe de ser enviado a todos los clientes registrados:

1. El *ServidorDeNotificaciones* pasa el *Evento* a la *TablaDeClientes*
2. La *TablaDeClientes* serializa el *Evento* creando con él un bloque de *bytes* que pueda ser enviado y posteriormente reconstruido por los clientes
3. La *TablaDeClientes* recorre la lista de *Clientes* al completo, enviando el mensaje de red a cada uno de ellos
4. Si el envío a un *Cliente* falla, esto significa que el mismo ha desconectado del servidor; en este caso, se cierra el *socket* y el *Cliente* es eliminado de la lista

Aunque el proceso es inicialmente sencillo, una consideración adicional requiere alterar el proceso indicado anteriormente volviéndolo bastante más complejo: **el envío de datos a un cliente remoto por TCP/IP es una operación bloqueante** (el hilo de ejecución principal no puede continuar hasta que se haya completado o haya fallado) **y no hay garantía de que vaya a resolverse de manera inmediata**.

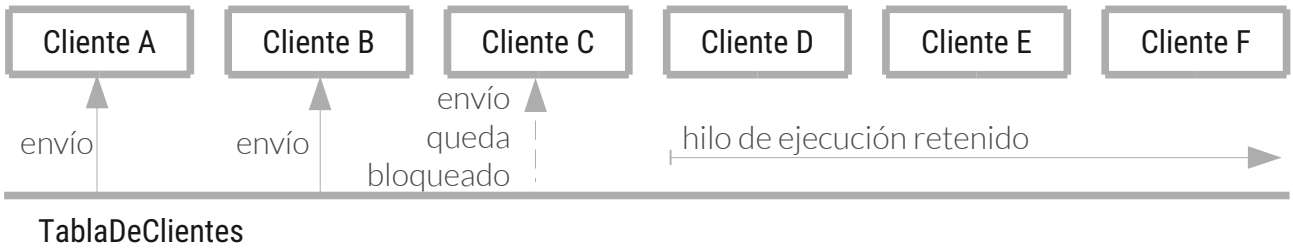

Ilustración 24: Retención del hilo de ejecución debido a envío bloqueado

Esto significa que para evitar que el servidor pueda quedar bloqueado durante más tiempo del necesario, reteniendo la cola de salida y el procesamiento de nuevos eventos, se hace necesario **procesar los envíos de manera asíncrona**.

Esto se implementa mediante el mecanismo de tareas asíncronas de la Librería Estándar C++ (*std::async*), que con la política de lanzamiento *launch::async* permite ejecutar una tarea (función) de forma asíncrona **y** concurrente al hilo principal.

Como el retorno de la llamada de envío es necesario para saber si el cliente ha desconectado, este se recoge mediante una variable futura (*std::future*), un tipo de datos genérico que puede utilizarse para la implementación del idioma *futuro-promesa* (uso de variables inicializadas de forma asíncrona en hilos de ejecución concurrentes) y recoger el valor de retorno de la tarea asíncrona.

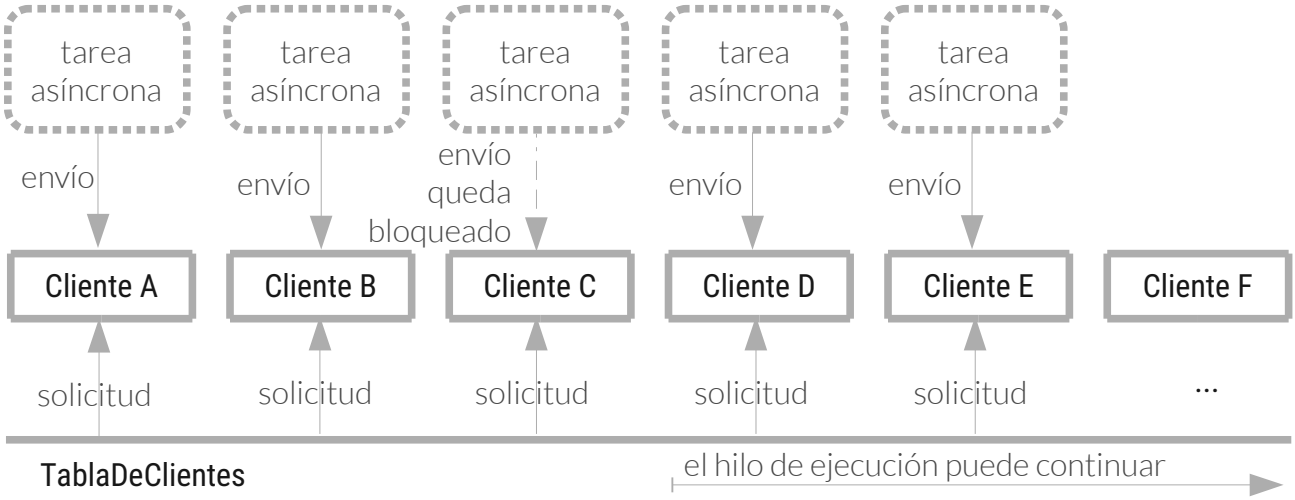

Ilustración 25: Uso de tareas asíncronas para evitar el bloqueo del hilo principal

No hay tampoco garantía de que un *Cliente* resuelva un envío antes de que se soliciten otros una vez que estos se resuelven de manera asíncrona. Por tanto, cada *Cliente* debe conservar una lista de valores de retorno futuros de envíos de datos solicitados. Antes de procesar un nuevo envío, puede recorrerse la lista para comprobar si alguno de los envíos ha fallado y por tanto procede cerrar el *socket* y eliminar el *Cliente* de la *TablaDeClientes*.

Después de estas alteraciones, la secuencia de actuación queda de la forma siguiente:

1. El *ServidorDeNotificaciones* pasa el *Evento* a la *TablaDeClientes*
2. La *TablaDeClientes* serializa el *Evento* creando con él un bloque de *bytes* que pueda ser enviado y posteriormente reconstruido por los clientes
3. La *TablaDeClientes* recorre la lista de *Clientes* al completo, solicitando el envío del mensaje de red para cada uno de ellos
4. Cada *Cliente* comprueba si alguna de las tareas de envío pendientes (en caso de haberlas) ha finalizado

5. En caso afirmativo, si alguna de estas ha fallado, devuelve el error a la *TablaDeClientes* para que elimine a dicho *Cliente*
6. Si la conexión está todavía viva, el *Cliente* lanza una tarea asíncrona para el envío del mensaje de red por el *socket* y almacena el retorno futuro de la misma en su lista de tareas pendientes

Este procedimiento asegura que el hilo de ejecución principal no quede nunca retenido por un envío pendiente garantizando asimismo que siempre que un cliente ha desconectado sea dado de baja y no se realicen más envíos a dicho cliente.

3.6 Implementación (cliente)

A continuación se proporciona una descripción textual de los aspectos fundamentales de la implementación en C++ de la aplicación servidora del sistema desarrollado.

3.6.1 Proceso

Al igual que el programa servidor, el programa cliente está implementado como un demonio que se ejecuta en segundo plano.

El procedimiento de creación del proceso demonio es análogo al presentado en 3.5.1 para el programa servidor.

3.6.2 Solicitud de servicio (conexión a servidores)

Intervienen: *ClienteDeNotificaciones*, *Servidor*

El *ClienteDeNotificaciones* es el encargado de solicitar la conexión a un *Servidor* mediante su dirección IPv4 o IPv6 y puerto TCP. Cada *Servidor* es una abstracción en la aplicación cliente de un servidor y contiene una conexión mediante un *socket* POSIX TCP/IP al mismo.

El programa es un cliente **multi-servidor**, es decir, que puede recibir servicio de múltiples servidores simultáneamente. Una vez establecida la conexión con un servidor, el cliente espera notificaciones de eventos de monitorización de ficheros de registro en dicho servidor hasta que su ejecución sea finalizada o el propio servidor deje de dar servicio.

La lectura de mensajes de red es bloqueante, lo que significa que para poder recibir servicio de varios servidores simultáneamente cada *Servidor* debe esperar mensajes entrantes en un hilo de ejecución paralelo propio. Esto se implementa haciendo uso de la Librería Estándar de C++ (*std::thread*).

Por tanto, el proceso de conexión consiste en:

1. Crear un *socket* de conexión a la dirección IP y puerto TCP apropiados
2. Crear un hilo de ejecución *std::thread* en paralelo al programa principal
3. En ese hilo, esperar mensajes de red provenientes del servidor al que se ha conectado

Este proceso se repite para cada servidor al que el usuario desee conectar, y a partir de ese momento todos ellos estarán en disposición de comenzar a transferir notificaciones de red para su procesado.

3.6.3 Procesamiento de eventos de notificación

Intervienen: *Servidor*, *CentroDeNotificaciones*, *Historial*, *Filtro*, *Evento*.

Cada vez que se recibe un nuevo evento de monitorización, el *Servidor* correspondiente hace una llamada al *CentroDeNotificaciones* y le pasa el objeto *Evento* correspondiente al mismo.

El *CentroDeNotificaciones* pasa ese *Evento* al *Historial* para que lo registre, lo evalúa haciendo uso del *Filtro*, y si lo pasa, muestra el *Evento* al usuario.

Como este proceso se efectúa desde diferentes hilos de ejecución (ya que cada *Servidor* actúa desde su propio hilo), se hace uso de *std::mutex* en la implementación para:

- Evitar condiciones de carrera en la escritura del fichero de *Historial*
- Evitar accesos concurrentes a *libnotify* (no es seguro en entornos multi-hilo)
- Garantizar que el orden en el que son registrados en el *Historial* es el mismo que el de notificación al usuario

Esta última razón es el motivo principal por el que los *mutex* se incluyen a este nivel. Se usan dos mutex diferentes para permitir a un segundo *Evento* registrarse en el *Historial* mientras el anterior es procesado en el *Filtro* con todas las *Reglas* que haya definido el usuario, pero ambos están entrelazados (no se desbloquea el primero hasta haber bloqueado el segundo) con objeto de garantizar la consistencia en cuanto al orden de aparición en el historial de la sesión y en la pantalla.

3.6.4 Registro de historial de sesión

Intervienen: *Historial*, *Evento*.

Cada vez que un *Evento* es procesado, si el uso del historial de sesión está activado por el usuario, el *Historial* simplemente lo escribe en el fichero correspondiente al historial de la sesión activa, añadiendo la fecha y hora y el servidor del que proviene para cada entrada.

El fichero de historial de sesión guarda sólo las notificaciones recibidas durante la sesión actual (es decir, desde que fue ejecutado), pero permite mantener un número especificado por el usuario (5 por defecto) de ficheros antiguos para su consulta posterior, añadiendo una extensión .1, .2, hasta .N a cada uno. Esto se implementa con un bucle que recorre los N ficheros comprobando si ya existen y renombrando con un número superior al que tenían, eliminando el último y creando uno nuevo limpio para la nueva sesión.

3.6.5 Filtrado de notificaciones al usuario

Intervienen: *Filtro*, *Regla*, *CondicionDeContenido*, *CondicionDeFichero*, *CondicionDeRemitente*.

El *Filtro* está implementado mediante comparaciones con expresiones regulares usando la Librería Estándar de C++ (*std::regex*).

Cada *Condicion* se compone de un *signo* (un booleano para indicar "igual" o "distinto" con *true* y *false* respectivamente) y una expresión regular escrita por el usuario. Con qué se compara la expresión depende del tipo de condición (*CondicionDeContenido* compara el texto de la entrada añadida al fichero de registro, *CondicionDeFichero* el nombre del fichero de registro, y *CondicionDeRemitente* la dirección del servidor desde el que es envía.

La función *evaluar* de una *Condicion* devuelve si es igual (==) el *signo* que la comparación de la expresión regular con el valor correspondiente al tipo de condición o no (en otras palabras, devuelve el resultado de la comparación de la expresión regular, o lo opuesto si el signo es *false* – distinto).

Una *Regla* es un vector de objetos *Condicion*, y se cumple (evalúa a *true*) sólo si todas las condiciones se cumplen (evalúan a *true*) para el *Evento* en cuestión: se recorre el vector evaluando cada *Condicion*, y tan pronto como una no se cumpla, se devuelve *false*. Si se recorre entero sin haber retornado, es que la regla se cumple y se devuelve *true*.

El *Filtro* se contiene un vector de objetos *Regla*. Si alguna de las reglas se cumple, el *Evento* debe ser omitido y no notificado. Se recorren todas las reglas y tan pronto se encuentre una que se cumpla, se devuelve *false* para indicar que el *Evento* no pasa el filtro. Si esto no ocurre, se retorna *true* y el *Evento* será notificado al usuario.

3.6.6 Notificación al usuario

Intervienen: *CentroDeNotificaciones*, *Evento*.

Para la notificación del *Evento* al usuario, se utiliza la librería *libnotify 0.7.6*. Su uso es sencillo. Al principio de la ejecución del programa, se inicializa *libnotify* para la aplicación. A partir de ese momento, cada vez que se quiera hacer una notificación, se crea un nuevo objeto de *Notification* y se construye la notificación modificando sus atributos con llamadas a varias funciones (construyendo la notificación de esa forma), y cuando se quiera mostrar, mediante la función *notifyNotification* se muestra en pantalla.

4 Planificación y presupuesto

4.1 Planificación

Para la planificación del proyecto se realiza una descomposición en tareas del proceso completo de desarrollo del mismo, con objeto de asignar a cada tarea una estimación del tiempo requerido para llevarla a cabo, y a partir de ésta, las fechas previstas para el inicio de cada una de forma que el proyecto completo sea desarrollado en los plazos requeridos.

La tarea de planificación se lleva a cabo durante la fase inicial del proyecto previa al comienzo de las tareas de desarrollo de éste, y se sigue hasta la conclusión del mismo. Sin embargo, a lo largo del proceso de desarrollo se producen inevitablemente reajustes y modificaciones a la misma, como la inclusión de nuevas tareas o redefinición de las previamente estipuladas en etapas posteriores a partir de la información obtenida conforme se van completando nuevas tareas, o reajustes en los periodos de tiempo asignado a cada tarea para adecuarlas a las necesidades reales conforme el proyecto va evolucionando.

La planificación presentada en este documento es la iteración final de la misma, tal y como resulta a fecha de fin de proyecto.

4.1.1 Tareas

El proceso de desarrollo completo se descompone en tareas con objetivos específicos y delimitadas.

La descomposición en tareas se realiza en dos niveles, con tareas correspondientes a distintas fases generales del proyecto (Estudio previo, Análisis, Diseño, etcétera) y tareas más específicas en las que se descompone cada una de ellas.

Para la mayoría de las fases del proyecto, además de la realización de las tareas de desarrollo correspondientes a dicha fase, se incluye una tarea correspondiente a la documentación del proceso para dicha fase. Esta tarea se desarrolla en paralelo a las tareas documentadas, ya que el proceso de documentación es continuo y se realiza según se progresa en el desarrollo de las tareas.

A continuación se listan las tareas consideradas en la planificación final del proyecto, incluyendo su Identificador (ID), Nombre, la Duración en días de la misma, y la fecha de comienzo de dicha tarea:

ID	Nombre	Duración	Comienzo
1	Estudio previo	9 días	Lunes 22/6/15
2	Definición del problema	1 día	Lunes 22/6/15
3	Viabilidad	1 día	Martes 23/6/15
4	Estado de la cuestión	5 días	Miércoles 24/6/15
5	Marco regulador	1 día	Miércoles 1/7/15
6	Planificación	1 día	Jueves 2/7/15
7	Documentación del estudio previo	9 días	Lunes 22/6/15
8	Análisis	11 días	Viernes 3/7/15
9	Especificación de requisitos	6 días	Viernes 3/7/15
10	Casos de uso	5 días	Lunes 13/7/15
11	Documentación del análisis	11 días	Viernes 3/7/15
12	Diseño	15 días	Lunes 20/7/15
13	Evaluación de alternativas de diseño	7 días	Lunes 20/7/15
14	Arquitectura del sistema	3 días	Miércoles 29/7/15
15	Modelado de clases	5 días	Lunes 3/8/15
16	Documentación del diseño	15 días	Lunes 20/7/15
17	Implementación	30 días	Lunes 10/8/15
18	Preparación del entorno de desarrollo	1 día	Lunes 10/8/15
19	Monitorización de ficheros	9 días	Martes 11/8/15
20	Notificación a usuarios	5 días	Lunes 24/8/15
21	Cliente-Servidor	4 días	Lunes 31/9/15
22	Servidor múltiples clientes	3 días	Viernes 4/9/15
23	Cliente múltiples servidores	3 días	Miércoles 9/9/15
24	Filtrado de notificaciones	3 días	Lunes 14/9/15
25	Historial de sesión	2 días	Jueves 17/9/15
26	Documentación de la implementación	30 días	Martes 11/8/15
27	Implantación y pruebas	5 días	Lunes 21/9/15
28	Uso en entorno de pruebas	3 días	Lunes 21/9/15
29	Manual de instalación	1 día	Jueves 24/9/15
30	Manual de usuario	1 día	Viernes 25/9/15

ID	Nombre	Duración	Comienzo
31	Redacción de conclusiones y trabajos futuros	2 días	Lunes 28/9/15
32	Revisión final	1 día	Miércoles 30/10/15

Tabla 51: Planificación de tareas del proyecto

4.1.2 Diagrama de Gantt

A partir de la descomposición del proyecto en las tareas presentadas anteriormente, la planificación del proyecto queda establecida. A continuación se muestra un diagrama de Gantt con la planificación resultante desarrollado para facilitar la visualización de la misma:

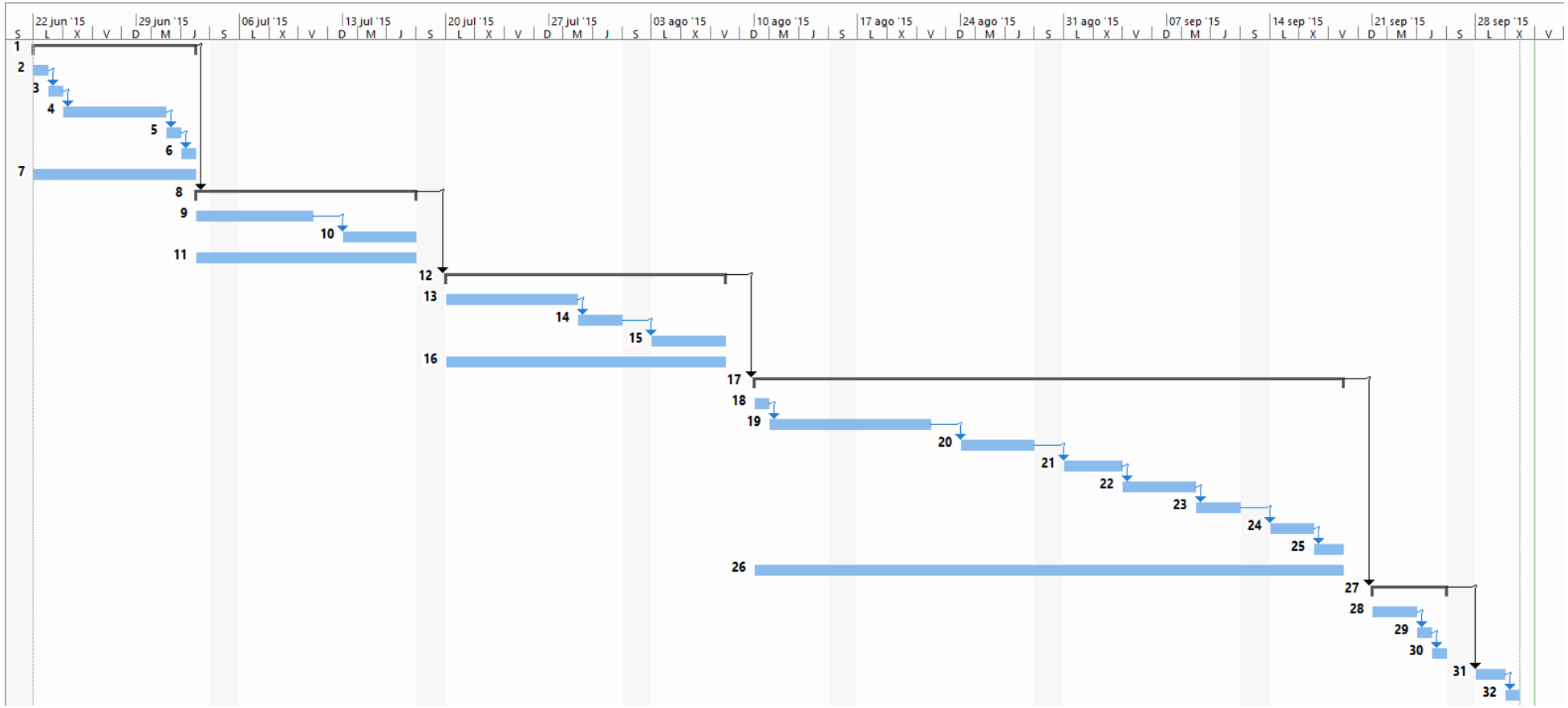

Ilustración 26: Diagrama de Gantt

4.2 Cálculo de presupuesto

Para el cálculo del presupuesto total necesario para el desarrollo de este proyecto se realiza un desglose por categorías indicando los costes totales incurridos para cada una y una especificación detallada del origen de cada uno de los costes.

Todos los costes especificados son costes finales, incluyendo IVA, otros impuestos aplicables, o costes derivados salvo que se especifique lo contrario.

4.2.1 Costes de personal

La categoría de Costes de personal incluye los costes de **mantenimiento de la plantilla** de personal asignada a las distintas tareas del proyecto. Los costes se calculan en base al *salario por hora* de cada uno de los roles participantes en el proceso de desarrollo, las tareas asignadas a cada uno y una jornada laboral de 8 horas diarias.

Para el desarrollo de este proyecto participan los siguientes roles de personal:

- **Analista de sistemas:** desarrolla las tareas de Estudio previo, Análisis, Conclusiones y Trabajos futuros y Revisión final (total: 184 horas).
- **Desarrollador de software:** lleva a cabo el Diseño del sistema (total: 120 horas).
- **Programador de aplicaciones:** realiza la Implementación y la Implantación y pruebas (total: 280 horas).

Los salarios medios para cada uno de estos roles se ha obtenido a partir de los datos recopilados por *tusalario.es*, parte del proyecto *WageIndicator Foundation*, a partir de 11.674 observaciones en España [46].

Rol	Salario	Tiempo asignado	Coste
Analista de sistemas	13,47 €/hora	184 horas	2.478,48 €
Desarrollador de software	12,28 €/hora	120 horas	1.473,60 €
Programador de aplicaciones	10,98 €/hora	280 horas	3.074,40 €
		Total bruto	7.026,48 €
		Seguridad social (23,6%)	1.637,52 €
		Total final	8.684,73 €

Tabla 52: Costes de personal

4.2.2 Costes de material

La categoría de Costes de material incluye los costes asociados a la **adquisición de activos fijos** necesarios para el desarrollo del proyecto, como materiales informáticos. El *periodo de depreciación* de dichos bienes se considera de 3 años (36 meses), en base al cuál se calcula el *coste atribuible* a la utilización de los mismos a lo largo del proyecto.

Activo	Coste por unidad	Depreciación	Utilización	Coste atribuible
PC de escritorio	1.600 €	1080 días	72 días	106,67 €
PC portátil	800 €	1080 días	72 días	53,33 €
Impresora	120 €	1080 días	72 días	8,00 €
2x pendrive	30 €	1080 días	72 días	4,00 €
			Coste total	172,00 €

Tabla 53: Costes de material

4.2.3 Costes de software

La categoría de Costes de software incluye los costes asociados a la adquisición de licencias de uso de software necesarios para el desarrollo del proyecto. La adquisición de licencias software se considera adquisición de activos fijos para su uso, por lo que está sujeta a los mismos cálculos de depreciación y coste atribuible de los Costes de material especificados en 4.2.2. El periodo de depreciación se calcula en función del tiempo aproximado que tarda en aparecer una nueva versión del software en cuestión.

Software	Coste por licencia	Depreciación	Utilización	Coste atribuible
Debian 8.1 Jessie	0 €	N/A	72 días	0 €
VS Code	0 €	N/A	72 días	0 €
LibreOffice 5.0	0 €	N/A	72 días	0 €
VMWare Player 7	134,95 €	360 días	72 días	26,99 €
MS Project 2013	1.369,00 €	1080 días	72 días	91,27 €
MS Windows 8.1 Pro	139,00 €	1080 días	72 días	9,27 €
			Coste total	127,53 €

Tabla 54: Costes de software

4.2.4 Costes de bienes fungibles

La categoría Costes de bienes fungibles incluye los costes asociados a la adquisición de bienes consumibles requeridos a lo largo del proceso de desarrollo del proyecto.

Consumible	Unidades/paquete	Coste/paquete	Cantidad	Coste total
Papel DinA4	500	3,00 €	1	3,00 €
Bolígrafos	10	4,00 €	1	4,00 €
Lápices HB	3	3,00 €	1	3,00 €
Discos DVD+R	10	6,75 €	1	6,75 €
Tinta impresora	1 tóner	49,00 €	1	49,00 €
			Coste total	65,75 €

Tabla 55: Costes de bienes fungibles

4.2.5 Costes indirectos

La categoría de Costes indirectos incluye los costes asociados al proceso productivo y el mantenimiento de uno o más productos, que no pueden por tanto asignarse a ninguna de las categorías anteriores de forma única.

Consumible	Unidad	Coste/unidad	Cantidad	Coste total
Electricidad	1 mes	variable	4	220,00 €
Línea telefónica	1 mes	17,40 €	4	69,60 €
Tarifa teléfono + Internet	1 mes	49,90 €	4	199,60 €
Transporte público	1 mes	63,70 €	4	254,80 €
			Coste total	744,00 €

Tabla 56: Costes indirectos

4.2.6 Cómputo global

Para el cómputo global de costes del proyecto, se suman los costes totales de cada una de las categorías detalladas con anterioridad y se obtiene el presupuesto necesario para completar el proyecto:

Categoría de costes	Coste total
Costes de personal	8.684,73 €
Costes de material	172,00 €
Costes de software	127,53 €
Costes de bienes fungibles	65,75 €
Costes indirectos	744,00 €
Coste total	9.794,01 €

Tabla 57: Cómputo global de presupuesto

5 Conclusiones y trabajos futuros

5.1 Conclusiones: sistema desarrollado

Recapitulando, el objetivo del proyecto tal y como se definió en el apartado 1.2 era realizar el Análisis, Diseño e Implementación de un prototipo funcional a modo de prueba de concepto de un sistema que permitiera monitorizar los ficheros de registro de GNU/Linux de forma que facilitase el conocimiento de los hechos en el momento en que se producen, pero hacerlo de una manera que no resultase intrusiva para el usuario, sin interrumpir su actividad normal. Más específicamente, se planteaba:

- Que el sistema desarrollado ofreciese información de actividad en el sistema monitorizado conforme esta se va produciendo
- Que lo hiciera de una forma no intrusiva y que no requiriese dedicación exclusiva
- Que pudiese ser utilizado a distancia desde un equipo diferente
- Que un usuario pudiese seguir múltiples equipos simultáneamente

¿Cumple con estos requisitos el sistema desarrollado?

A través del uso de *inotify* se obtienen eventos de sistema de ficheros en el momento en que se producen, sin *polling*, y esto permite un seguimiento y procesamiento de los ficheros de registro eficiente. El sistema desarrollado obtiene la información necesaria de los ficheros cuando son modificados, y opera de forma estable en el contexto de los ficheros de registro, incluso cuando el SO realiza tareas de mantenimiento (como la rotación de los ficheros de registro).

Las notificaciones de escritorio mostradas a través de *libnotify* son de por sí poco intrusivas: según el demonio servidor de notificaciones instalado en el sistema en que se use, pueden aparecer en el borde inferior de la pantalla, cerca de la esquina superior derecha… siempre en regiones periféricas de la misma, y si son ignoradas, desaparecen por si solas en pocos segundos.

Para evitar un exceso de notificaciones intrascendentes, la función de filtrado implementada supone una opción muy flexible ya que muchas de las entradas introducidas en los ficheros de registro son acciones repetitivas y que por tanto tienen patrones fáciles de representar con expresiones regulares. Al ser las reglas definidas por el usuario, es posible adaptar la frecuencia de las notificaciones recibidas a las preferencias individuales o a cada sistema.

En general, el sistema puede operar de forma muy transparente (incluso automatizando su lanzamiento, una vez se han escrito los ficheros de configuración) e integrada con otros

mensajes de sistema.

Por último, la arquitectura Cliente-Servidor implementada con *sockets* TCP/IP con múltiples Clientes suscritos simultáneamente a un Servidor, y la posibilidad de suscribir un Cliente a múltiples Servidores a la vez, cubre las necesidades de acceso remoto a través de red local o Internet, y lo hace de forma eficiente gracias al uso del modelo de Cliente-Servidor tipo push, con un modelo de comunicación análogo al del patrón Publicación-Suscripción.

5.2 Conclusiones: proceso de desarrollo

El proceso de desarrollo seguido funcionó, en líneas generales, correctamente.

Todas las tareas establecidas en la planificación pudieron ser finalmente ser completadas con éxito, y el trabajo previo de investigación y análisis sobre la problemática planteada y las tecnologías y soluciones disponibles para acometer los objetivos proporcionó una base sólida para que el resto del proceso de desarrollo transcurriese de forma fluida.

Debido a la dimensión manejable y características del proyecto, la metodología en cascada resultó en general acertada, y sólo ocasionalmente fue necesario volver a alguna etapa anterior y realizar modificaciones.

La mayor parte de las dificultades encontradas durante el proceso de desarrollo tuvieron que ver con subestimar consistentemente el tiempo requerido para algunas de las tareas, particularmente la investigación previa, la implementación (mucho más lenta de lo esperado debido principalmente a la proporción de elementos de bajo nivel en la misma, y la falta de familiaridad con algunas técnicas y tecnologías empleadas), y la elaboración de la documentación.

Estas dificultades en la estimación del tiempo requerido para completar las diferentes tareas tuvieron como resultado el retraso progresivo de los hitos marcados para la progresión del proyecto.

5.3 Conclusiones personales

A título personal, este proyecto me ha ofrecido varias lecturas interesantes.

Por un lado ejemplifica cómo en ocasiones es suficiente con cambiar la perspectiva, **dar un enfoque distinto a una solución ya existente** para obtener una nueva mejor, o simplemente diferente (capaz de resolver problemas de otras características). Los ficheros de registro, un concepto omnipresente entre la mayoría (si no totalidad) de sistemas operativos, son un viejo conocido de cualquier administrador de sistemas: cuando ha ocurrido un problema, los *logs* del sistema son la primera herramienta a la que recurrir para conocer la causa. Pero ¿y si en lugar de verlos *después*, los vemos *durante*? De pronto tenemos un medio de monitorización pasiva de actividad del sistema a partir de una potente infraestructura ya existente con información de todo tipo, y sólo necesitamos para ello una herramienta que nos proporcione la nueva forma de visualizarlo.

Por supuesto y como ya hemos visto, este proyecto no es el primero en plantear esta idea: herramientas como *swatch*, o incluso simples scripts basados en *tail*/*tailf*, se han valido con anterioridad de este concepto para proporcionar una solución similar.

¿Dónde fallan, o al menos son mejorables, estas soluciones en mi opinión? **Usabilidad**, uno de los grandes protagonistas en el mundo del desarrollo de tecnologías y foco principal de algunas de las empresas tecnológicas más exitosas a nivel internacional. En gran medida, la usabilidad depende de cómo integrar las nuevas funcionalidades a nuestra rutina habitual sin interferir con ésta. Y el hecho es que dicha usabilidad puede provenir no sólo del diseño de la interfaz de usuario (el sistema de notificación al usuario en este proyecto) sino de cualquier otro aspecto del sistema como conjunto. ¿Qué pasa si no puedo/quiero estar delante del PC que quiero monitorizar? **El diseño del sistema como sistema distribuido tiene mucho que ver con la usabilidad resultante en el producto final**. En este sentido una línea de desarrollo futuro para este sistema que me parece particularmente interesante es la implementación de un medio de notificación para dispositivos móviles: ¿qué pasa si no puedo/quiero estar delante de un ordenador en absoluto?

Mirando atrás, durante el desarrollo de este proyecto he podido ahondar en áreas como el desarrollo de sistemas distribuidos, varias técnicas de ejecución concurrente y asíncrona, lidiar con las dificultades de la planificación y estimación de trabajo para un proyecto de desarrollo de software, e incluso aprovechar para cumplir algunos propósitos personales como familiarizarme con las novedades de la especificación de C++11, que llevaba un tiempo queriendo explorar, todo ello cosas que espero me resulten de gran utilidad de cara al futuro.

5.4 Trabajos futuros

Utilizando el proyecto desarrollado como punto de partida, se ofrece a continuación una serie de sugerencias sobre cómo podría expandirse el trabajo realizado:

- Realizar una **evaluación de rendimiento y estudio de escalabilidad**, procediendo a la implantación del sistema en un entorno multi-usuario relativamente grande, midiendo el impacto en cuanto a tráfico de red o procesamiento del uso del sistema en un entorno real.
- Añadir **soporte para sistemas de notificación adicionales**, ya que pese a que el uso de D-Bus y la *Desktop Notifications Specification* (y por extensión *libnotify*) están bastante extendidas entre las distintas distribuciones GNU/Linux, el soporte adicional de otros sistemas de notificación podría permitir hacer uso de otras plataformas para la monitorización.
- Relacionado con el punto anterior, un **sistema de notificación para dispositivos móviles** resulta un concepto muy interesante. Ya que el motivo fundamental de la implementación de un sistema de monitorización remota es poder desarrollar otras actividades no localizadas en el equipo o equipos monitorizados, la notificación a dispositivos móviles es el paso siguiente: no requerir al usuario estar frente a un PC para proseguir con el proceso de monitorización. El uso de notificaciones desde aplicaciones móviles está muy extendido hoy en día, y en combinación con las posibilidades de filtrado de notificaciones para reducir la cantidad de avisos recibidos, podría ser utilizado de forma realista.
- Implementar un **sistema de autenticación y conexión segura** que permita restringir el acceso a los servidores de eventos sobre los ficheros de registro a usuarios conocidos e identificables, y proteger así la información contenida en los ficheros de registro de los equipos monitorizados.
- Implementar una **interfaz gráfica de usuario** para el cliente, que permitiese acceder al proceso cliente ejecutado en segundo plano para comprobar el estado de las conexiones, conectar/desconectar servidores, cambiar la configuración y las reglas de filtrado o consultar el historial de sesión de forma centralizada, intuitiva y sin necesidad de detener y reiniciar el proceso cliente.

Anexos

Anexo I: Acrónimos

Acrónimo	Significado
API	Application Programming Interface
E/S	Entrada y Salida
FHS	Filesystem Hierarchy Standard
GNU	GNU's Not UNIX
GPL	General Public License
I/O	Input and Output
IP	Internet Protocol (como en *IP address*, *Internet Protocol address*)
LGPL	Lesser General Public License
LSB	Linux Standard Base
MIT	Massachusetts Institute of Technologies
POSIX	Portable Operating System Interface
SO	Sistema Operativo
SW	Software
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

ANEXO II: MANUAL DE INSTALACIÓN

Anexo II: Manual de instalación

El desarrollo y pruebas del sistema se han llevado a cabo en GNU/Linux Debian 8.1 (Jessie) por lo que se recomienda utilizar al menos la versión proporcionada en los repositorios de Debian correspondientes a dicha versión para todos los programas y herramientas utilizados para el proceso de instalación.

La aplicación servidora requiere GNU/Linux con *kernel* 2.6.13 o superior para su ejecución. La aplicación cliente requiere GNU/Linux. Para conectar de forma remota con la aplicación servidora, se requiere una conexión de red o Internet con un puerto TCP conocido y disponible habilitado (comprobar la situación de cortafuegos o *NAT*).

Mapa de archivos y directorios del proyecto

Cada una de las dos aplicaciones se encuentra en un directorio independiente:

- **lognotifyserv** contiene los archivos correspondientes a la aplicación servidora.
- **lognotifycli** contiene los archivos correspondientes a la aplicación cliente.

En cada uno de estos directorios se encuentra la misma estructura de directorios y ficheros:

- **src** contiene los archivos de código fuente C++ con extensión .cpp y .h.
- **obj** contiene los archivos de código objeto generados durante la compilación.
- **bin** contiene los archivos binarios ejecutables generados durante el vinculado.
- **doc** contiene los archivos html de documentación generada por *doxygen*.

En el directorio raíz de cada aplicación (*lognotifyserv* y *lognotifycli*) se esncuentra un archivo *Makefile* para la compilación automática de la aplicación correspondiente.

Dependencias

La aplicación cliente requiere 2 librerías externas para compilar:

- **glib-2.0**, parte de la librería GTK+.
- **libnotify**.

Si no se dispone de ellas, pueden ser instaladas en Debian desde la línea de comandos con:

apt-get install libgtk2.0-dev libnotify-dev

ANEXO II: MANUAL DE INSTALACIÓN

Compilación

Para compilar cualquiera de las dos aplicaciones mediante la aplicación *make* y *g++*:

1. Situarse en el directorio correspondiente a la aplicación que se desea compilar (*lognotifyserv* o *lognotifycli*).
2. Teclear *make*.
3. El ejecutable generado se encuentra en el directorio *bin* dentro de esa misma ubicación.

Cada una de las dos aplicaciones (cliente y servidor) se compila por separado de la otra.

La aplicación está escrita en C++11, por lo que se requiere una versión lo suficientemente reciente del compilador *g++* como para que tenga soporte completo. Se recomienda utilizar una versión lo más reciente posible. El sistema ha sido desarrollado y probado con *g++ 4.9.2*. Se recomienda por tanto utilizar dicha versión o una más reciente.

Ficheros de configuración

Al ejecutar tanto el cliente como el servidor, ambos esperan encontrar ciertos ficheros de configuración necesarios para la ejecución en un directorio por defecto. Este directorio por defecto es **$HOME/.lognotify**. Es posible especificar un directorio diferente al ejecutar la aplicación, por lo que estos ficheros pueden ser ubicados en cualquier otra localización si así se desea.

En dicho directorio, la aplicación servidora buscará el archivo **ficheros**, donde el usuario o administrador deberá especificar la lista de ficheros que desea monitorizar (más información en el Anexo III: Manual de usuario).

En el mismo directorio, la aplicación cliente buscará los archivos **servidores** y **filtro**, que contienen un listado de las direcciones de los servidores a los que conectar y las reglas de filtrado de notificaciones respectivamente (más información en el Anexo III: Manual de usuario).

Es conveniente crear estos ficheros si se desea dejar el sistema preparado para su funcionamiento.

ANEXO III: MANUAL DE USUARIO

Anexo III: Manual de usuario

Para utilizar el sistema es necesario ejecutar al menos una instancia de cada de la aplicación servidora y la aplicación cliente. Ambas pueden ejecutarse en el mismo equipo o en equipos diferentes. La aplicación servidora (**lognotifyserv**) debe de ser ejecutada en el/los equipo/s cuyos ficheros de registro se desee monitorizar. La aplicación cliente (**lognotifycli**) debe de ser ejecutada en el/los equipos en los que se desee recibir notificaciones de escritorio avisando de los eventos ocurridos en los equipos monitorizados.

Ejecución del servidor

Para poder acceder a los ficheros de registro del sistema contenidos por defecto en /var/log es necesario disponer de privilegios suficientes, por lo que es probable que deba ejecutarse con elevación de privilegios mediante el comando *sudo*.

La aplicación **lognotifyserv** acepta uno o más de los siguientes parámetros:

- **-p ####** indica el puerto TCP en el que acepta conexiones entrantes de clientes. Este parámetro es necesario para la ejecución. #### es el número de puerto, siendo un número positivo menor o igual que 65535. Se recomienda utilizar un número de puerto en el rango 49152 a 65535, correspondiente a Puertos Privados o Dinámicos según las asignaciones IANA. Es importante comprobar el estado del puerto escogido, especialmente en relación al uso de cortafuegos o NAT.
- **-d** hace que el programa se ejecute como demonio.
- **-f ruta** indica una ruta alternativa para los ficheros de configuración (por defecto es **$HOME/.lognotify**).
- **-w ruta** indica una ruta alternativa donde se ubican los ficheros de registro del sistema (por defecto es **/var/log**)
- **-h** despliega la ayuda en pantalla para la ejecución del programa.

La aplicación **lognotifyserv** utiliza un fichero de configuración llamado **ficheros**. Dicho archivo se utiliza para indicar al programa los ficheros que debe monitorizar. Cada fichero debe de ser escrito en una línea diferente como ruta relativa desde el directorio de ficheros de registro del sistema (por defecto /var/log). Esto significa que para los ficheros que su ubiquen directamente bajo ese directorio solo hay que escribir el nombre (por ejemplo: auth.log) y para aquellos que se encuentren en un subdirectorio, la ruta desde ese punto (por ejemplo: application1/file.log).

ANEXO III: MANUAL DE USUARIO

Ejecución del cliente

La aplicación **lognotifycli** acepta uno o más de los siguientes parámetros:

- **-d** hace que el programa se ejecute como demonio.
- **-f ruta** indica una ruta alternativa para los ficheros de configuración (por defecto es **$HOME/.lognotify**).
- **-t ####** especifica un tiempo en ms (milisegundos) de permanencia en pantalla para las notificaciones.
- **-s ####** especifica el número de sesiones de historial antiguas que deben conservarse sin eliminar para consulta posterior (por defecto 5).
- **-r** indica al programa que debe mostrar la ruta completa del fichero de registro en la cabecera de las notificaciones (por defecto sólo muestra el nombre del mismo).
- **-o** indica al programa que debe mostrar la dirección IP del servidor de origen de una notificación al final de la misma (por defecto no la muestra).
- **-h** despliega la ayuda en pantalla para la ejecución del programa.

La aplicación **lognotifycli** utiliza dos ficheros de configuración llamados **servidores** y **filtro**.

El fichero **servidores** se utiliza para indicar al programa los servidores a los que debe conectar. Cada servidor debe ser escrito en una línea diferente con el formato xxx.xxx.xxx.xxx/#### con la dirección IPv4 antes de la barra, y el puerto de escucha del mismo después (por ejemplo: 192.168.1.44/50000).

El fichero **filtro** se utiliza para definir las reglas de filtrado de notificaciones que permiten omitir algunas de ellas:

Cada regla tiene varias condiciones. Una nueva regla comienza cuando una nueva línea empieza con la palabra **regla**. El resto de la línea es ignorado, se recomienda utilizarla para nombrar o describir la regla con objeto de recordar posteriormente lo que hacía (por ejemplo: regla No mostrar notificaciones de auth.log).

Cada nueva línea hasta la siguiente regla, es una condición de dicha regla. Hay 3 tipos de condiciones: de **origen**, de **fichero** y de **contenido**. Cada una empieza la línea con la palabra correspondiente. A continuación se indica si la expresión debe de ser igual para que se cumpla la condición, o lo contrario (**=**, o **!=**). Todos los caracteres que sigan a estos símbolos, se considerarán parte de la expresión regular a comparar para decidir si la condición se cumple. El programa reconoce expresiones regulares en formato ECMAScript. Si la condición

ANEXO III: MANUAL DE USUARIO

es de tipo **origen**, se compara con la dirección IP del servidor de origen; si es de tipo **fichero**, se compara con el nombre del fichero que origina la notificación; por último si es de tipo **contenido**, se compara con el contenido textual de la entrada que se ha añadido al fichero de registro causando la notificación.

Para que una regla se cumpla, deben de cumplirse **todas** las condiciones. Si una condición no se cumple, la regla entera no se cumple para la notificación en cuestión.

Si **una** de las reglas especificadas se cumple, la notificación **es omitida**. Para que una notificación se muestre, no debe haber ninguna regla definida que se dispare para dicha notificación.

Por ejemplo, si tenemos el fichero de reglas:

regla No mostrar notificaciones de casa salvo auth.log

origen=192\.168\.1\.34

fichero!=auth\.log

regla No mostrar notificaciones de auth.log

fichero=auth\.log

Y recibimos un mensaje de notificación del PC con IP 192.168.1.34 sobre el fichero auth.log, la primera regla no se cumplirá, porque aunque la IP coincide, el fichero *no* es distinto de auth.log. Por tanto esa regla no omitirá la notificación. Sin embargo, la segunda sí se disparará (ya que se cumple para toda notificación sobre auth.log), por lo que la notificación no será mostrada de todas formas.

NOTA: como se ve en el fichero de ejemplo, se ha utilizado el carácter de escape \ antes de cada punto (.). Es importante recordar que las expresiones utilizadas para las comparaciones son expresiones regulares en formato ECMAScript, y en este el punto es un carácter reservado que indica “cualquier carácter excepto terminadores de línea”. Es importante también darse cuenta de que el uso de expresiones regulares ECMAScript permite crear reglas de filtrado mucho más potentes y versátiles que las anteriores, que se han mantenido lo más sencillas posible para dar a entender el formato de definición de las mismas con facilidad.

Referencias

[1] Linux Foundation. *Filesystem Hierarchy Standard*, [en línea]. Reference specification, version 3.0 (2015). Dirección URL: <http://refspecs.linuxfoundation.org/FHS_3.0/fhs/index.html>. [Consulta: 8 septiembre 2015].

[2] Linux Foundation. *FHS home page*, [en línea]. LSB Workgroup. Dirección URL: <http://www.linuxfoundation.org/collaborate/workgroups/lsb/fhs>. [Consulta: 8 septiembre 2015].

[3] Unsigned Integer Limited. *DistroWatch.com*, [en línea]. Dirección URL: <http://distrowatch.com>. [Consulta: 8 septiembre 2015].

[4] Linux Foundation. *Certification Management System: certified products*, [en línea]. LSB Workgroup. Dirección URL: <https://www.linuxbase.org/lsb-cert/productdir.php?by_lsb>. [Consulta: 8 septiembre 2015].

[5] Debian. *Debian Policy Manual*, [en línea]. Versión 3.9.6.1 (22-11-2014). Dirección URL: <https://www.debian.org/doc/debian-policy/>. [Consulta: 8 septiembre 2015].

[6] Fedora. *Overview of File System Hierarchy Standard*, [en línea]. Fedora Documentation. Dirección URL: <https://docs.fedoraproject.org/en-US/Fedora/14/html/Storage_Administration_Guide/s1-filesystem-fhs.html>. [Consulta: 8 septiembre 2015].

[7] OpenSUSE. *Concepts directory structure*, [en línea]. Documentation wiki. Dirección URL: <https://en.opensuse.org/Concepts_directory_structure>. [Consulta: 8 septiembre 2015].

[8] Ubuntu Documentation. *LinuxLogFiles*, [en línea]. Community Help Wiki. Dirección URL: <https://help.ubuntu.com/community/LinuxLogFiles>. [Consulta: 8 septiembre 2015].

[9] MintGuide. *What is the purpose of the folders in the directory structure of Linux Mint?*, [en línea]. Dirección URL: <http://mintguide.org/system/234-what-is-the-purpose-of-the-folders-in-the-directory-structure-of-linux-mint.html>. [Consulta: 8 septiembre 2015].

[10] GoboLinux. *GoboLinux at a glance*, [en línea]. Dirección URL: <http://www.gobolinux.org/?page=at_a_glance>. [Consulta: 8 septiembre 2015].

[11] Infodrom. *sysklogd*, [en línea]. Syslogd and klogd home page. Dirección URL: <http://www.infodrom.org/projects/sysklogd/>. [Consulta: 8 septiembre 2015].

[12] Gerards, Rainer. *rsyslogd*, [en línea]. Sourceforge. Dirección URL: <http://sourceforge.net/projects/rsyslog/>. [Consulta: 8 septiembre 2015].

[13] BalaBit IT Security. *syslog-ng*, [en línea]. syslog-ng home page. Dirección URL: <https://www.balabit.com/network-security/syslog-ng>. [Consulta: 8 septiembre

2015].

[14] Debian. *Man Page Lookup* [en línea]. (Última modificación: 3 septiembre 2015). Dirección URL: <http://manpages.debian.org/cgi-bin/man.cgi>. [Consulta: 14 septiembre 2015].

[15] GNU. "*tail.c*" [en línea]. *GNU coreutils*. Savannah GNU git repositories. Dirección URL: <http://git.savannah.gnu.org/cgit/coreutils.git/tree/src/tail.c>. [Consulta: 14 septiembre 2015].

[16] Atkins, Todd. *Simple Log Watcher* [en línea]. Sourceforge. Dirección URL: <http://sourceforge.net/projects/swatch/>. [Consulta: 8 septiembre 2015].

[17] Bauer, Kirk. *Logwatch* [en línea]. Sourceforge. Dirección URL: <http://sourceforge.net/projects/logwatch/>. [Consulta: 9 septiembre 2015].

[18] The Open Group. *POSIX.1-2008* [en línea]. 2013 Edition. The Open Group & IEEE. Dirección URL: <http://pubs.opengroup.org/onlinepubs/9699919799/>. [Consulta: 17 septiembre 2015].

[19] Debian. *"fcntl Man Page"* [en línea]. *Man Page Lookup*. Dirección URL: <http://manpages.debian.org/cgi-bin/man.cgi?query=fcntl&apropos=0&sektion=0&manpath=Debian+8+jessie&format=html&locale=en>. [Consulta: 17 septiembre 2015].

[20] Debian. *"inotify Man Page"* [en línea]. *Man Page Lookup*. Dirección URL: <http://manpages.debian.org/cgi-bin/man.cgi?query=inotify&apropos=0&sektion=0&manpath=Debian+8+jessie&format=html&locale=en>. [Consulta: 17 septiembre 2015].

[21] Growl Team. *Growl* [en línea]. Dirección URL: <http://growl.info/>. [Consulta: 18 septiembre 2015].

[22] Yasushiro Matsumoto. *Growl For Linux* [en línea]. Github. Dirección URL: <http://mattn.github.io/growl-for-linux/>. [Consulta: 18 septiembre 2015].

[23] Growl for Windows. *Growl for Windows* [en línea]. Dirección URL: <http://www.growlforwindows.com/gfw/>. [Consulta: 18 septiembre 2015].

[24] Daryl Ginn. *Growl Notification Style* [en línea]. Dribbble. Dirección URL: <https://dribbble.com/shots/304135-Growl-Notification-Style>. [Consulta: 18 septiembre 2015].

[25] freedesktop.org. *D-Bus* [en línea]. (Última modificación 25 agosto 2015). Dirección URL: <http://www.freedesktop.org/wiki/Software/dbus/>. [Consulta: 18 septiembre 2015].

[26] freedesktop.org. *freedesktop.org homepage* [en línea]. (Última modificación 17 mayo 2013). Dirección URL: <http://www.freedesktop.org/wiki/>. [Consulta: 18 septiembre 2015].

[27] The GNOME Project. *Desktop Notifications Specification* [en línea]. Versión 1.2. Dirección URL: <https://developer.gnome.org/notification-spec/>. [Consulta: 18 septiembre 2015].

[28] The GNOME Project. *Libnotify Reference Manual* [en línea]. Dirección URL: <https://developer.gnome.org/libnotify/>. [Consulta: 18 septiembre 2015].

[29] Google Developers. *chrome.notifications* [en línea]. Google. Dirección URL: <https://developer.chrome.com/apps/notifications>. [Consulta: 18 septiembre 2015]

[30] Google Developers. *What Are Chrome Apps?* [en línea]. Google. Dirección URL: <https://developer.chrome.com/apps/about_apps>. [Consulta: 18 septiembre 2015].

[31] Google Developers. *Rich Notifications* [en línea]. Google. Dirección URL: <https://developer.chrome.com/apps/richNotifications>. [Consulta: 18 septiembre 2015].

[32] Microsoft. "*Windows Sockets 2*" [en línea]. *Windows Dev Center*. MSDN. Dirección URL:<https://msdn.microsoft.com/en-us/library/windows/desktop/ms740673(v=vs.85).aspx>. [Consulta: 17 septiembre 2015].

[33] Christopher Kohlhoff. *Boost.Asio* [en línea]. Boost.org. (Última modificación 4 agosto 2015). Dirección URL: <http://www.boost.org/doc/libs/1_59_0/doc/html/boost_asio.html>. [Consulta: 17 septiembre 2015].

[34] Boost.org. *Boost C++ Libraries* [en línea]. Dirección URL: <http://www.boost.org/>. [Consulta: 17 septiembre 2015].

[35] Lee Salzman. *ENet Reliable UDP networking library* [en línea]. (Última modificación 30 abril 2015). Dirección URL: <http://enet.bespin.org/index.html>. [Consulta: 17 septiembre 2015].

[36] Jenkins Software. *RakNet* [en línea]. Dirección URL: <http://www.jenkinssoftware.com/index.html>. [Consulta: 17 septiembre 2015].

[37] Google Developers. *Cloud Messaging* [en línea]. Google. (Última modificación 26 agosto 2015). Dirección URL: <https://developers.google.com/cloud-messaging/>. [Consulta: 18 septiembre 2015].

[38] España. Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal. *Boletín Oficial del Estado*. 14 de diciembre de 1999, número 298.

[39] España. Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia. *Boletín Oficial del Estado*. 22 de abril de 1996, número 97.

[40] The GTK+ Project. *What is GTK+, and how can I use it?* [en línea]. Dirección URL: <http://www.gtk.org/>. [Consulta: 23 septiembre 2015].

[41] The GNU Project. *GNU Lesser General Public License, version 2.1* [en línea]. Dirección URL: <http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html>. [Consulta: 23 septiembre 2015].

[42] The GNOME Project. *copyright* [en línea]. Fichero de texto contenido en el paquete descargable de la librería Libnotify. URL: <http://ftp.gnome.org/pub/GNOME/sources/libnotify/0.7/>. [Consulta: 23 septiembre 2015].

[43] The GNU Project. *GNU General Public License* [en línea]. Dirección URL: <http://www.gnu.org/licenses/gpl.html>. [Consulta: 23 septiembre 2015].

[44] Open Source Iniciative. *The MIT License (MIT)* [en línea]. Dirección URL: <https://opensource.org/licenses/MIT>. [Consulta: 23 septiembre 2015].

[45] Open Standards. *C++ Standards* [en línea]. ISO/IEC. (publicación: 1 septiembre 2011) (última modificación: 5 mayo 2014). Dirección URL: <http://www.open-std.org/jtc1/sc22/wg21/docs/standards#14882>. [Consulta: 29 septiembre 2015].

[46] WageIndicator Foundation. *tusalario.es* [en línea]. Dirección URL: <http://www.tusalario.es/main>. [Consulta: 28 septiembre 2015].